





2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein

Módulo 1 - Introducción a Desarrollo Web y Aplicaciones

Índice







2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein

GIT y GITHUB

¿Qué es la programación?

Es el proceso utilizado para idear y ordenar las acciones necesarias para realizar un proyecto, preparar ciertas máquinas o aparatos para que empiecen a funcionar en el momento y en la forma deseados o elaborar programas para su empleo en computadoras.

En la actualidad, la noción de programación se encuentra muy asociada a la creación de aplicaciones de informática y videojuegos. En este sentido, es el proceso por el cual una persona desarrolla un programa, valiéndose de una herramienta que le permite escribir el código (el cual puede estar en uno o varios lenguajes, como C ++, Java y Python, entre otros) y de otra que sea capaz de “traducirlo” a lo que se conoce como lenguaje de máquina, que puede "comprender" el microprocesador.

¿Qué es un algoritmo?

Un algoritmo informático es un conjunto de instrucciones definidas, ordenadas y acotadas para resolver un problema o realizar una tarea. En programación, supone el paso previo a ponerse a escribir el código. Primero debemos encontrar la solución al problema (definir el algoritmo informático), para luego, a través del código, poder indicarle a la máquina qué acciones queremos que lleve a cabo. De este modo, un programa informático no sería más que un conjunto de algoritmos ordenados y codificados en un lenguaje de programación para poder ser ejecutados en un ordenador.

Habilidades digitales para programadores

¿A qué nos referimos con habilidades digitales?

Las Tecnologías de información y comunicación (TIC) se han convertido en herramientas que intervienen en la mayor parte de las actividades laborales, académicas y recreativas de la vida actual. En nuestros días nos enfrentamos cotidianamente a situaciones de interacción social mediadas por las TIC: relaciones sociales, transacciones comerciales, trámites, consulta, intercambio y producción de información, situaciones de estudio, recreación, etc.

Saber moverse en este mundo con alto uso de tecnologías de información y participar en los variados tipos de intercambios mediados por las TIC puede definirse como estar integrado a la cultura digital. Para esto es necesario contar con habilidades digitales. Entendemos por habilidades digitales el conjunto de saberes (saber hacer y saber sobre el hacer) relacionados con el uso de herramientas de comunicación, acceso, procesamiento y producción de la información.

De manera resumida, las habilidades digitales no sólo remiten a aspectos operacionales, sino a aspectos formales, como cuál es la mejor solución a determinado problema, o la forma adecuada de encarar una dificultad tecnológica. Por ahora, haremos foco en habilidades de búsqueda.

Uso de Keywords:

Primero y más importante es pensar bien los términos de búsqueda que vamos a usar. Cuando usamos algún motor de búsqueda tenemos algunos indicadores de eficiencia. Por ejemplo, generar menor cantidad de resultados de búsqueda implica que nuestra búsqueda ha sido más precisa. Pero menos no siempre es mejor. También podemos obtener menos resultados si no expresamos de manera correcta lo que estamos buscando.

Por ejemplo, nos encontramos aprendiendo a usar Githhub, y queremos saber cómo clonar un repositorio, pero no de la rama principal, sino de una rama en específico:

1. **Primera opción:** “cómo clonar un repositorio de github de una rama secundaria” (49,400 resultados)

2. **Segunda opción:** “Clone secondary Branch github” (295,000 resultados)

3. **Tercera opción:** “github clonar una rama específica” (43,600 resultados)

4. **Cuarta opción:** “github clonar rama” (12,100,000 resultados)

Como vemos, la cantidad de resultados puede ser un indicador, pero no es el único… Por supuesto si utilizamos términos en inglés tendremos más resultados. Algunos consejos que te podemos dar son:

1. Analiza tu problema, planea el trabajo de búsqueda.
2. Piensa y anota qué posibles términos de búsqueda usar y ve ampliando la lista.
3. Considera qué herramientas de búsqueda emplear, usa varias herramientas.
4. Examina cómo se utilizan las herramientas de búsqueda, aprende a usarlas.
5. Usa sistemas de búsqueda avanzada (campos, frases, limitaciones, combinación…).
6. Busca ordenada y sistemáticamente, en varios pasos lógicos, sin precipitación.
7. Guarda resultados provisionales, analízalos después, conserva los definitivos, etc.
8. Valora críticamente los resultados, piensa si son relevantes, de un nivel adecuado.

Uso de Operadores de búsqueda

Los operadores de búsqueda son una serie de símbolos y comandos que sirven para acotar los resultados de las búsquedas que realizas. Algunos de ellos incluyen símbolos como el de puntuación, que Google siempre ignora en el caso de que no pertenezcan a un operador concreto.

Son como filtros añadidos, sólo que en vez de estar en las opciones que ves debajo de la barra de búsqueda de Google cuando te muestra los resultados tienes que incluirlos en la propia búsqueda acompañando al término que quieras encontrar. Hay incluso algunos que sirven como enlace para poder utilizar más de uno a la vez.

La forma en la que funcionan es muy sencilla. Lo único que tienes que hacer es escribir un texto en la barra de búsqueda de Google en el que se incluya el operador que quieres utilizar. Habrá veces en las que este operador estará entre comandos de búsqueda o al final de cada uno, dependiendo de lo que quieras hacer con él.







2021 Año de homenaje al premio Nobel de medicina Dr. César Milstein

Operadores de búsqueda de Google

OPERADOR	EJEMPLO	QUÉ HACE
OR	Pelota OR palo OR paso	Te muestra resultados que contengan cualquiera de las palabras que hayas incluido.
AND	JavaScript and Sintaxis	Busca páginas que incluya los dos términos especificados.
" "	"Github flow" o "Subversion"	Te muestra resultados donde aparece el término o los términos exactos que hayas añadido entre los ".
-	Fullstack -MEAN	Te muestra resultados donde se excluya la palabra que hayas puesto detrás del -.
*****	"SCRUM *meeting"	Un comodín que puede coincidir con cualquier palabra en la búsqueda.
#..#	Celular 20000..50000 pesos	Te muestra resultados donde donde se añade un intervalo de números que tú especificas.
( )	("redes sociales" OR "plataformas sociales") -Twitter	Te permite combinar operadores. En el ejemplo buscarás redes sociales o plataformas sociales, pero excluyendo Twitter de los resultados.
AROUND	Trucos around(3) Instagram	Resultados donde aparecen las dos palabras especificadas, pero con el número que determines de términos entre ellas.
EN	300 dólares en euros	Sirve para convertir unidades de un mismo tipo de medida.
MAP	map:BuenosAires	La búsqueda te devuelve resultados con mapas del sitio donde le digas.
DEFINE	define:Lunfardo	Busca la definición de una palabra que no conozcas
SITE	Cursos site:www.argentina.gob.ar	Te busca los resultados dentro de una web que hayas especificado.
INFO	info:www.argentina.gob.ar/	Te muestra resultados donde se ofrezca información sobre una página web.
RELATED	related:www.argentina.gob.ar/	Te muestra en los resultados otras páginas similares a la que has escrito.
LINK	Teléfonos link:https://developer.mozilla.org/	Te muestra en los resultados páginas que tienen enlaces a la web que hayas especificado.
CACHÉ	cache:https://developer.mozilla.org/	Te muestra la copia de la página que hay en el caché de Google.
FILETYPE	filetype:pdf presupuestos 2021	Busca resultados que contengan archivos con el formato que hayas especificado
ALLINTEXT O INTEXT	allintext:"desarrollador fullstack"	Encuentra páginas que incluyan en su texto algunos o todos los términos que hayas incluido en el comando.
ALLINTITLE O INTITLE	allintitle:recursos desarrollador web	Te muestra páginas que tengan algunos o todos los términos que hayas incluido en el comando en su título
INURL O ALLINURL	inurl:"apple iphone" allinurl:"apple sfera"	Te muestra páginas con algunos o todos los términos que hayas incluido en el comando

OPERADOR	EJEMPLO	QUÉ HACE
ALLINANCHOR O INANCHOR	allinanchor:"desarrollador fullstack"	Resultados con páginas donde se incluya un enlace con un texto anclado donde se incluya uno o varios términos especificados.
STOCKS	stocks:Facebook	Busca el estado actual de la empresa que busques en bolsa.
WEATHER	weather:Rosario,ar	El tiempo en la ciudad elegida. Mira que después del nombre de la ciudad puedes poner una coma y el país para ser más concreto.
TIME	time:Nueva York	Te muestra la hora en la localidad que decidas.
MOVIE	movie:Avengers	Te muestra resultados relacionados con una película que establezcas.
@	@ArgentinaPrograma	Busca etiquetas sociales asociadas con Twitter.
#	#ArgentinaPrograma	Busca términos publicados con hastags en redes sociales que tengan sistema de hashtags.

Alternativas Motores de Búsqueda

Dónde buscar

Empecemos por un pequeño detalle. Conviene tener presente que Google no es el único buscador generalista, es decir, orientado a rastrear y localizar sitios y páginas web de cualquier clase. Hay otros, como, por ejemplo:

1. Yahoo search
2. Bing
3. Exalead
4. Startpage
5. Duckduckgo
6. Ask
7. Search encrypt
8. Brave

Google es el buscador más conocido y utilizado. Quizá es el que más información rastrea de internet y también el que en líneas generales mejor lo hace. Pero no hay que menospreciar la capacidad de sus rivales de encontrar resultados preferibles para algunos problemas, ni las prestaciones especiales de algunos, como Exalead o StartPage.

Pedir a veces una segunda opinión, tener otros buscadores de reserva a la hora de explorar la web, es buena idea. Puede resultarnos de utilidad.

También en caso de necesitarlo en algún momento, hay motores de búsqueda especializados en contenido académico, por ejemplo:

- Google Scholar
- Microsoft Academic
- Base
- World Wide Science

Herramientas de desarrollador

¿Cuáles son las herramientas de desarrollo del navegador?

Todos los navegadores web modernos incluyen un potente conjunto de herramientas para desarrolladores. Estas herramientas hacen una variedad de cosas, desde inspeccionar HTML, CSS y JavaScript actualmente cargados, hasta mostrar qué activos ha solicitado la página y cuánto tiempo tardaron en cargarse. Vamos a explicar cómo utilizar las funciones básicas de las herramientas de desarrollo del navegador.

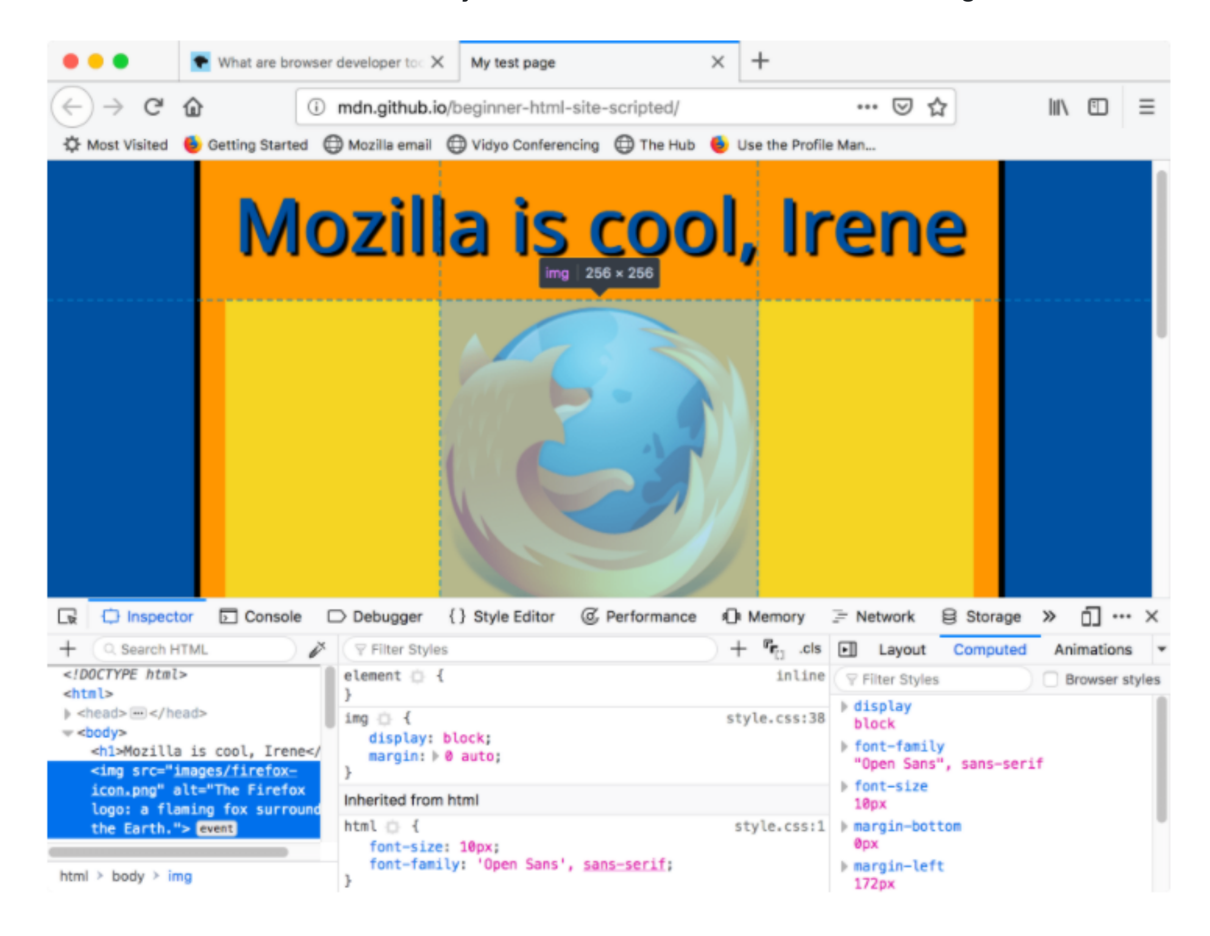
Cómo abrir devtools en tu navegador

Las herramientas para desarrolladores (devtools) viven dentro de tu navegador en una subventana que se ve más o menos así, dependiendo del navegador que estés utilizando.

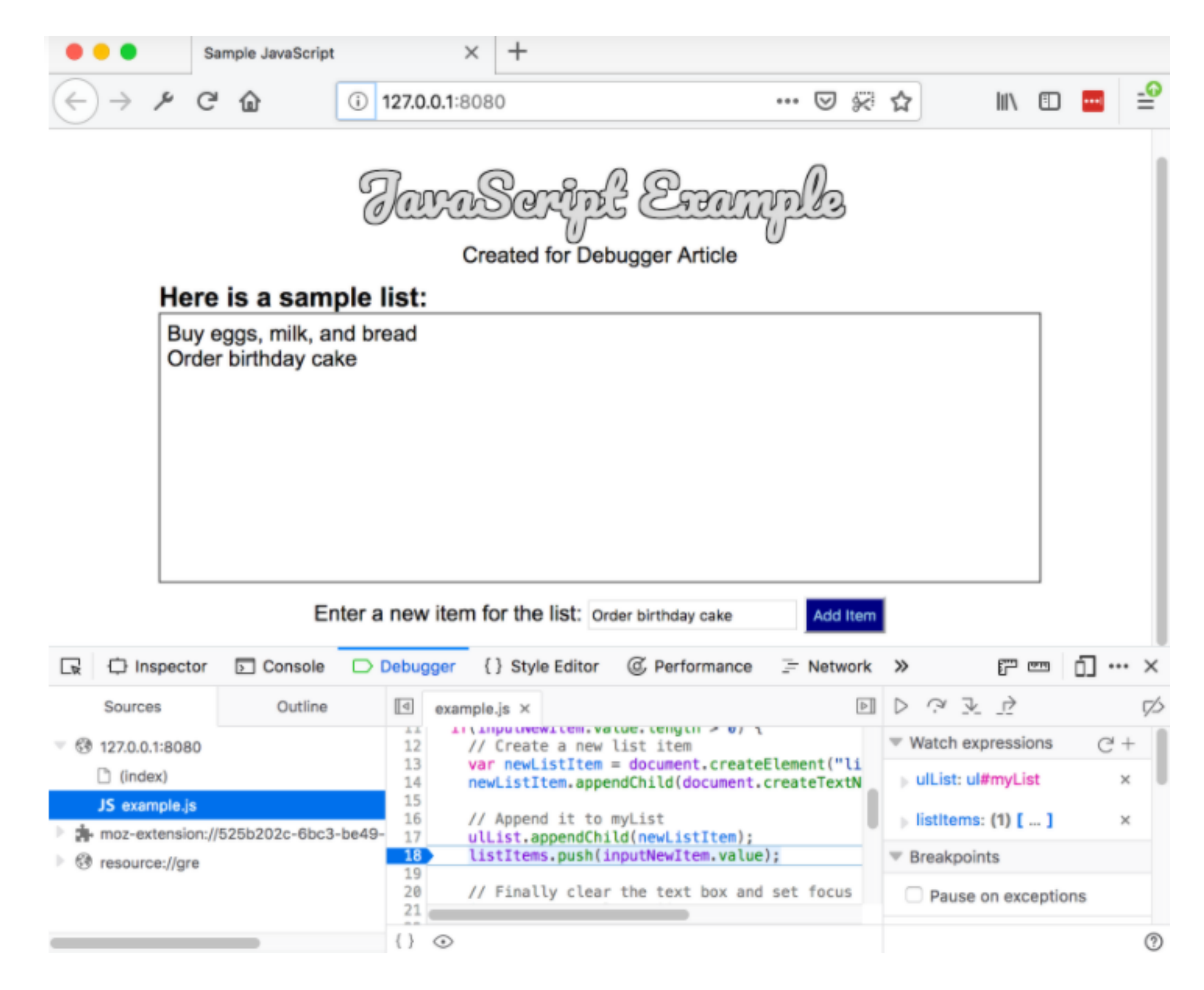
¿Cómo la levantas? Existen tres distintas maneras:

- ***Teclado:*** *Ctrl*+*Mayús*+*I*, excepto en
  - **Internet Explorer y Edge:** F12
  - **macOS:** *⌘*+*⌥*+*I*
- ***Barra de menú:***
  - **Firefox**: Menú ☰ ➢ *Desarrollador web* ➢ *Alternar herramientas,* o ➢ *Herramientas* ➢ *Alternar herramientas del desarrollador web*
  - **Chrome:** *Más herramientas* ➢ *Herramientas del desarrollador*
  - **Safari:** *Desarrollador* ➢ *Mostrar el inspector web.* Si no puedes ver el menú *Desarrollar*, ve a *Safari* ➢ *Preferencias* ➢ *Avanzado* y marca la casilla de verificación *Mostrar menú desarrollador en la barra de menú*.
  - **Opera**: *Desarrollador* ➢ *Herramientas para desarrolladores*
- ***Menú contextual:*** Presiona y mantén presionado / haz clic con el botón derecho en un elemento en una página web (Ctrl-clic en Mac) y elige *Inspeccionar elemento* en el menú contextual que aparece. (*Una ventaja adicional:* este método, inmediatamente resalta el código del elemento en el que hiciste clic con el botón derecho).

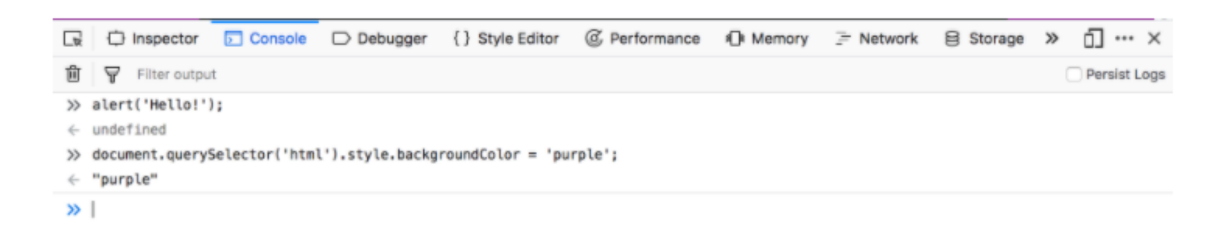
Las herramientas del desarrollador, generalmente se abren de forma predeterminada en el inspector. Esta herramienta muestra cómo se ve el HTML en tu página en tiempo de ejecución, así como qué CSS se aplica a cada elemento de la página. También te permite modificar instantáneamente el HTML y CSS y ver los resultados de tus cambios reflejados en vivo en la ventana del navegador.



Además, podemos encontrar el depurador de JavaScript, el cual te permite observar el valor de las variables y establecer puntos de interrupción, lugares en tu código en los que deseas pausar la ejecución e identificar los problemas que impiden que tu código se ejecute correctamente.



Otro componente importante es la consola de JavaScript que es una herramienta increíblemente útil para depurar JavaScript que no funciona como se esperaba. Te permite ejecutar líneas de JavaScript en la página actualmente cargada en el navegador e informa los errores encontrados cuando el navegador intenta ejecutar tu código.



Herramientas para programadores

Empecemos por algunos puntos que nos van a ser útiles…

Traductor:
El clásico es GoogleTranslate. Revisemos algunos detalles de su uso.
El Traductor de Google es un sistema multilingüe de traducción automática, desarrollado y proporcionado por Google, para traducir texto, voz, imágenes o video en tiempo real de un idioma a otro. El Traductor de Google posee la capacidad de traducir en más de 100 idiomas en distintos niveles, el sistema provee un servicio gratuito y diariamente es utilizado por más de 200 millones de personas.

Para poder utilizar el traductor debes activar el plugin del navegador de Chrome (https://chrome.google.com/webstore/detail/google-translate/aapbdbdomjkkjkaonfhkkikfgjllcleb) o Firefox (https://addons.mozilla.org/es/firefox/addon/to-google-translate/) También dando click derecho en la página en la que te encuentres y luego "Traducir a español".
Otra opción recomendable es DeepL que te permite traducir texto con mayor calidad, pero no puedes hacerlo con una página completa.
https://www.deepl.com/translator

Licenciamiento y versionado:

Existen varios criterios de clasificación para dividir los tipos de licencias de software. La tabla 1 facilita la comprensión e incorpora ejemplos para cada tipo de licencia. En el caso del software propietario, las licencias de software van a depender del titular de los derechos de autor del software en cuestión, que normalmente va a ser quien lo crea o quien lo ofrece. Veamos una pequeña descripción de las más importantes:

Licencias de software Shareware

Corresponden a un tipo de distribución de aplicaciones que consiste en liberar gratuitamente una versión con funcionamiento limitado. Esa limitación puede ser temporal (después de determinada cantidad de días deja de operar), por funciones (desde el comienzo, o a partir de determinado momento, hay funciones que el programa deja de realizar) o una combinación de las mencionadas (el programa empieza con todas sus funciones y deja de realizar algunas al cabo de cierto tiempo).

Licencias de software libre

En el caso del software libre, si bien cada desarrollador puede utilizar la licencia que desee, como ocurre con el software propietario, está más extendida la práctica de licenciar un software libre bajo determinadas licencias creadas principalmente por organizaciones, como la licencia General Public License (GNU GPL), creada y promovida por la Free Software Foundation, la licencia Apache, la licencia Mozilla Public License, creada y promovida por la Mozilla Foundation y usada en su producto más conocido, el navegador web Mozilla Firefox, entre otras.

Otros tipos de licencias de software

Además de esta distinción entre licencias de software libre y licencias de software propietario, existen otras clasificaciones de las licencias:
• Según el grado de libertad de uso que se le entrega al licenciatario.
• Según el grado de estandarización de los términos de la licencia.
• Según la forma de celebración del contrato.







• Según el grado de estandarización, licencias de software genérico (empaquetado) con contratos de adhesión y licencias de software personalizado. Las licencias otorgadas mediante contratos de adhesión se llaman también licencias shrink-wrap.
• Según la forma de celebración del contrato, distingue entre licencias celebradas por escrito; licencias celebradas por otros medios válidos de expresión del consentimiento y licencias celebradas por medios electrónicos. Hay que señalar que las licencias celebradas por escrito sería un contrato consensual, que se perfecciona por el solo consentimiento de las partes.

Suele ocurrir con el software libre que, al ser un conjunto de aportes de distintos desarrolladores, cada “parte” de un software tenga licencias distintas. Así ocurre, por ejemplo, con el sistema operativo móvil Android y la iniciativa de código abierto que lo construye y mantiene, el Android Open Source Project (AOSP), en castellano Proyecto de Código Abierto de Android. En su sitio web, el AOSP recomienda el uso de la licencia Apache para liberar el software relacionado con Android. Sin embargo, hay partes de Android que se liberan bajo otras licencias, como la licencia GPL para el núcleo (kernel) del sistema, puesto que es el núcleo del sistema operativo Linux, utilizado no solamente como base de Android sino también como base para sistemas operativos de escritorio, de servidores, de sistemas embebidos (por ejemplo, cajeros automáticos o navegadores GPS), que está liberado bajo la licencia GPL.

Licencias de Uso:

LAS COMBINACIONES, **DE UN VISTAZO**

MÁS PERMISIVA

SI VEO ESTA LICENCIA...		YO PUEDO ...	SI...
PUBLIC DOMAIN	**Dominio Público** (CC0): Europeana, Figshare, Open Goldberg V.	• Compartir • Copiar • Remezclar • Ganar dinero	• Menciono al autor (en algunas jurisdicciones)
CC BY	**Reconocimiento** (by): PLOS, Saylor.org.	• Compartir • Remezclar • Ganar dinero	• Menciono al autor
CC BY SA	**Reconocimiento – CompartirIgual** (by-sa): Wikipedia, Wikimedia, Arduino, P2PU	• Compartir • Remezclar • Ganar dinero	• Menciono al autor • Mantengo la misma licencia (by-sa)
CC BY ND	**Reconocimiento – SinObraDerivada** (by-nd): Drupal, Behance, GNU, Free Software Foundation.	• Compartir • Ganar dinero	• Menciono al autor • No hago remezclas
CC BY NC	**Reconocimiento – NoComercial** (by-nc): Brooklyn Museum, Wired.com Photography	• Compartir • Remezclar	• Menciono al autor • No gano dinero
CC BY NC SA	**Reconocimiento – NoComercial – CompartirIgual** (by-nc-sa): MIT Open CourseWare	• Compartir • Remezclar	• Menciono al autor • No gano dinero • Mantengo la misma licencia (by-nc-sa)
CC BY NC ND	**Reconocimiento – NoComercial – SinObraDerivada** (by-nc-nd): Videos TED Talks, Propublica	• Compartir	• Menciono al autor • No hago remezclas • No gano dinero • Mantengo la misma licencia (by-nc-nd)

MÁS RESTRICTIVA

Sitios de proyectos y comunidades:

Todas las tecnologías tienen y cuentan con documentación oficial, para acceder a ella es importante que conozcas el origen de ella. Aquí te damos algunos ejemplos con las tecnologías que trabajaras en este curso y que te ayudarán durante toda tu vida como programador:

• Documentación oficial HTML - https://developer.mozilla.org/es/docs/Web/HTML
• Documentación oficial CSS - https://developer.mozilla.org/es/docs/Web/CSS
• Documentación oficial JavaScript - https://developer.mozilla.org/es/docs/Web/JavaScript/Reference
• Documentación oficial Angular - https://angular.io/docs
• Documentación oficial Java - https://docs.oracle.com/javaee/7/index.html
• Documentación oficial SpringBoot - https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
• Página Oracle University (Java) - https://education.oracle.com/software/java/pFamily_48

Stackoverflow:
La descripción que hace la propia página web del servicio nos parece lo suficientemente concreta y precisa, por lo que no hace falta añadir demasiado a la descripción.
"Stack Overflow es una comunidad abierta para cualquiera que codifique. Lo ayudamos a obtener respuestas a sus preguntas de codificación más difíciles, compartir conocimientos con sus compañeros de trabajo en privado y encontrar el próximo trabajo de sus sueños"

https://es.stackoverflow.com/

Si eres un desarrollador, puedes acceder a diferentes funciones y características de gran utilidad. Entre ellas, se encuentran las siguientes:

- Obtener respuestas a más de 16.5 millones de preguntas
- Retribuir el apoyo compartiendo conocimiento con otros desarrolladores
- Compartir información de manera privada con compañeros de trabajo
- Encontrar trabajo adecuado a través de listados de alta calidad, filtrando mediante título, tecnología, salario, ubicación y otras variables

Glosario de recursos útiles

Cursos de Programación (y más):

- ✓ Udacity: https://www.udacity.com/
- ✓ Coursera: https://es.coursera.org/
- ✓ Canal Youtube MIT: https://www.youtube.com/c/mitocw
- ✓ EdX: https://www.edx.org/es/

Herramientas CSS:

- ✓ CSS Gradient: https://cssgradient.io/
- ✓ CSS Layout: https://csslayout.io/
- ✓ Getwaves: https://getwaves.io/
- ✓ WebGradients: https://webgradients.com/
- ✓ Cubic-bezier: https://cubic-bezier.com/#.17,.67,.83,.67
- ✓ CSS3: https://www.css3.me/
- ✓ CSS-Minifier: https://cssminifier.com/







2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein

Herramientas BootStrap:
- ✓ Icons.getbootstrap: https://icons.getbootstrap.com/
- ✓ Themes.getbootstrap: https://themes.getbootstrap.com/
- ✓ Bootswatch: https://bootswatch.com/
- ✓ Getbootstrap: https://getbootstrap.com/

Herramientas HTML:
- ✓ HTML-minifier: https://html-minifier.com/
- ✓ HTML ColorCode: https://htmlcolorcodes.com/es/
- ✓ W3schools: https://www.w3schools.com/

Herramientas e Imágenes Alta resolución:
- ✓ Freepik: https://www.freepik.es/
- ✓ Pexels: https://www.pexels.com/es-es/
- ✓ MotionArray: https://motionarray.com/
- ✓ UnDraw: https://undraw.co/
- ✓ UnSplash: https://unsplash.com/
- ✓ FreeImages: https://www.freeimages.com/es
- ✓ Pixabay: https://pixabay.com/es/
- ✓ Thenounproject: https://thenounproject.com/
- ✓ Tinypng: https://tinypng.com/
- ✓ Manypixels: https://www.manypixels.co/
- ✓ Humaans: https://www.humaaans.com/
- ✓ Uigradients: https://uigradients.com/#CocoaaIce
- ✓ Flaticon: https://www.flaticon.es/
- ✓ Boxicons: https://boxicons.com/

Herramientas FrontEnd:
- ✓ Codepen: https://codepen.io/
- ✓ Modernizr: https://modernizr.com/
- ✓ BrowserShots: https://browsershots.org/
- ✓ CDNJS: https://cdnjs.com/
- ✓ Waybackmachine: https://archive.org/web/
- ✓ Dummyimage: https://dummyimage.com/
- ✓ Colorzilla: https://www.colorzilla.com/
- ✓ Caniuse: https://caniuse.com/
- ✓ Spritecow: http://www.spritecow.com/
- ✓ Figma: https://www.figma.com/
- ✓ JavasScript-minifier: https://www.minifier.org/

Fuentes:
- ✓ fonts.google: https://fonts.google.com/
- ✓ Es.lipsum: https://es.lipsum.com/
- ✓ FontAwesome: https://fontawesome.com/







2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein

Editores de código:

- ✓ VScode: https://code.visualstudio.com/

VCS:

- ✓ Subversión: https://subversion.apache.org/
- ✓ Mercurial: https://www.mercurial-scm.org/
- ✓ Git: https://git-scm.com/
- ✓ Github: https://github.com/
- ✓ Gitlab: https://about.gitlab.com/
- ✓ Bitbucket: https://bitbucket.org/

Programas para VCS:

- ✓ Github-desktop: https://desktop.github.com/
- ✓ Git-kraken: https://www.gitkraken.com/

Sistema de persistencia:

- ✓ SQLite: https://www.sqlite.org/index.html
- ✓ MariaDb: https://mariadb.org/

- ✓ MongoDb: https://www.mongodb.com/es
- ✓ Oracle: https://www.oracle.com/ar/index.html
- ✓ PostgreSQL: https://www.postgresql.org/
- ✓ Redis: https://redis.io/
- ✓ MySQL: https://www.mysql.com/
- ✓ DynamoDb: https://aws.amazon.com/es/dynamodb/
- ✓ AmazonRelationDataBaseService: https://aws.amazon.com/es/rds/

Navegadores:

- ✓ Chrome: https://www.google.com/intl/es-419/chrome/
- ✓ Brave: https://brave.com/es/
- ✓ Chrome Canary: https://www.google.com/intl/es-419/chrome/canary/
- ✓ Chromium: https://www.chromium.org/
- ✓ Firefox Developer Edition: https://www.mozilla.org/es-AR/firefox/developer/

Servicios web:

- ✓ Rest API
- ✓ Graphql API – Serverless Firebase
- ✓ AWS Amplify
- ✓ Cloudinary

Desarrollo API:

- ✓ Postman: https://www.postman.com/product/graphql-client/
- ✓ Rest Client de VSCode: https://marketplace.visualstudio.com/items?itemName=humao.rest-client

Servicios cloud:

- ✓ Azure: https://azure.microsoft.com/es-es/
- ✓ Amazon Web Services: https://aws.amazon.com/es/
- ✓ GoogleCloud: https://cloud.google.com/
- ✓ Docker - para empaquetar: https://www.docker.com/

CMS:

- ✓ Wordpress: https://wordpress.com/es/
- ✓ Woocommerce: https://woocommerce.com/
- ✓ Magento: https://magento.com/
- ✓ Blogger: https://www.blogger.com/about/
- ✓ Ghost: https://ghost.org/

Framework de Backend:

- ✓ GO (Gorilla, Buffalo, goji)
- ✓ Pyhton (django, flask)
- ✓ Typescript/JavaScript/Node (Loopback, Nest, Next.js, Nuxts.js)

Servers:

- ✓ Nginx: https://www.nginx.com/
- ✓ Apache: https://www.apache.org/
- ✓ Windowsiis: https://www.iis.net/
- ✓ Tomcat: http://tomcat.apache.org/

Mobiledev:

- ✓ ionic (angular react vim)
- ✓ react native + expo, flutter

Breve reseña

A partir del desarrollo de ARPANET en 1969 empieza un crecimiento vertiginoso del uso de la internet. En 1990 Tim Berners-Lee creó la **WWW** , la "WorldWideWeb" que realizó la primera conexión desde un navegador a un servidor web mientras trabajaba en el CERN desarrollando así, las tres tecnologías fundamentales de la web que son:

- **HTML** (lenguaje de marcado de hipertexto). Lenguaje de marcado o etiquetado que se emplea para escribir los documentos o páginas web.
- **URL** (localizador de recursos universal). El localizador de recursos uniforme, sistema de localización o direccionamiento de los documentos web.
- **HTTP** (Protocolo de transferencia de hipertexto) El lenguaje con el que se comunica el navegador con el servidor web y el que se emplea para transmitir los documentos web.

Se trata entonces de una arquitectura cliente-servidor en la que cada dispositivo electrónico en la red (internet , intranet o extranet) actúa como cliente o servidor lo que implica la comunicación entre procesos que hacen peticiones (clientes) y

procesos que responden a esas peticiones (servidores). Esta comunicación es posible gracias al protocolo HTTP.



Arquitectura cliente / servidor básico

En 1994 (1 de octubre) Tim Berners-Lee abandona el CERN y funda la W3C, en inglés, "World Wide Web Consortium", organismo internacional que propone recomendaciones y estándares web que aseguran el crecimiento de la World Wide Web.

Haciendo clic aquí podrás ver la evolución de la web. Mencionaremos los hitos más relevantes:

- En 1991 surge **HTTP** definido como "protocolo de red para sistemas de información hipermedia distribuidos".
- Muy próximo aparece **HTML** 1, es el lenguaje de marcado predominante de las páginas web.
- En 1995, Netscape creó **JavaScript**, un lenguaje de secuencias de comandos basado en prototipos y "orientado a objetos". El objetivo de este lenguaje de programación fue darle capacidad de ejecución al cliente de esta arquitectura web, es decir, al navegador.
- En 1998 aparecen las hojas de estilo, en su versión 2. Se denominaron **CSS**, del inglés "Cascading Style Sheets", que es un lenguaje de hojas de estilo empleado para describir la semántica de presentación de un documento, en este caso un documento web

Arquitectura de las aplicaciones Web

Las aplicaciones web se basan en una arquitectura cliente / servidor. Es decir que, por un lado, está el cliente (navegador) y por otro lado el servidor. Existen diferentes variantes de la arquitectura básica según como se implementa, pero e s importante mencionar que en la tecnología la mayoría de las estructuras están compuestas por capas.

A continuación, enumeramos algunas de las arquitecturas más comunes:

- Servidor web + base de datos en un mismo servidor (2 niveles o capas). En este caso el servidor gestiona tanto la lógica de negocio como la lógica de los datos y los datos.



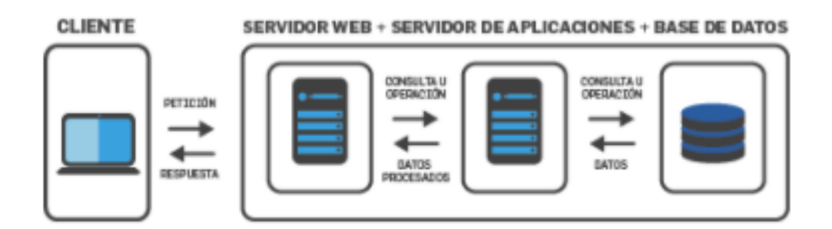
Servidor web + base de datos en un mismo servidor

- Servidor web y de datos separados (3 niveles). En este caso se separa la lógica de negocio a la de datos en diferentes servidores.



Servidor web y de datos separados

- Servidor web + servidor de aplicaciones + base de datos en un mismo servidor (2 niveles).



Servidor web + servidor de aplicaciones + base de datos en un mismo servidor

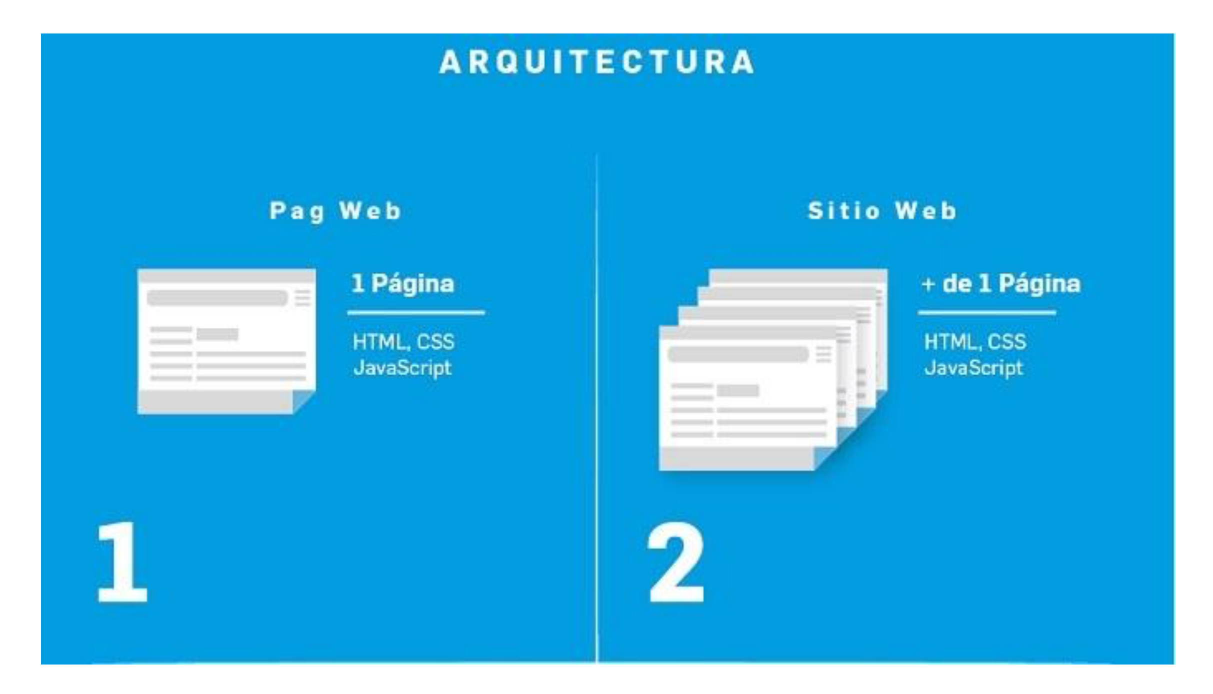
- Servidor de aplicaciones + base de datos en un mismo servidor (3 niveles).
- Servidor de aplicaciones, base de datos y servicios de aplicaciones en diferentes separados en diferentes servidores (4 niveles).

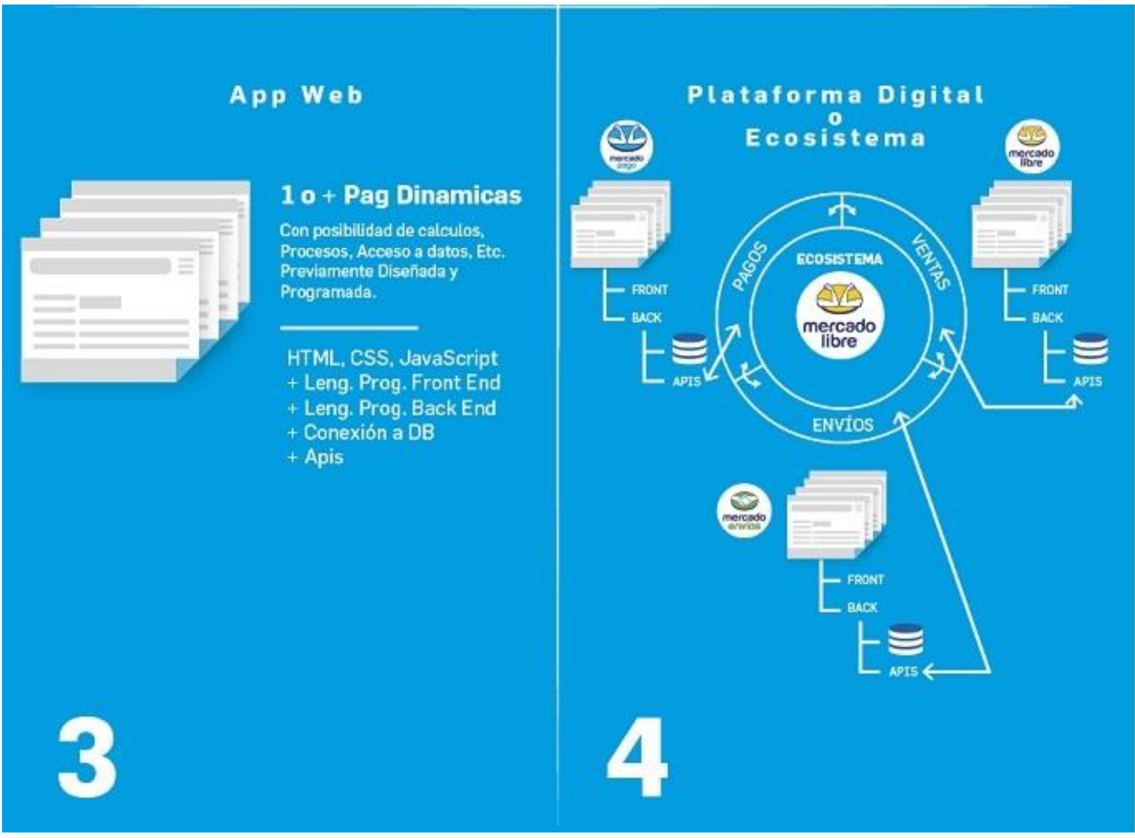
Como vemos, la arquitectura web tiene un patrón de diseño en capas (arquitectura distribuida), y cada capa puede estar en un servidor diferente y aun así se pueden interconectar. El objetivo de separar las distintas funcionalidades en distintos servidores es aumentar la escalabilidad y el rendimiento. Por ejemplo, el servidor web al ofrecer servicios de http deberá tener una buena conexión a internet mientras que el servidor de base de datos requiere tener una buena capacidad de almacenamiento y procesamiento.

Un grupo de páginas web dinámicas se conceptualiza como Front End (pensadas para que el cliente acceda) y el otro grupo de páginas dinámicas web como Back End (pensadas para el procesamiento y acceso a datos), además de que la base de datos puede existir en otro Servidor. Esto da lugar a un concepto muy

importante en POO el DESACOPLAMIENTO, en este caso el diseño del Front End, Back End y Base de Datos puede desacoplarse.

Ahora veamos distintos escenarios de arquitecturas web, debemos tener presente que nos referimos a como se dividen o distribuyen los distintos componentes de una aplicación en una red, en algunos casos dependiendo su complejidad del desarrollo necesita ser distribuida en varios servidores. La complejidad es determinada por las necesidades de la problemática que se pretende resolver o el diseño de la aplicación. Hagamos un repaso por las distintas arquitecturas que se nos pueden presentar en la siguiente imagen:





Analicemos juntos con más detalles cada escenario planteado.

- **En el escenario 1:** Vemos una página web con contenido estática, es decir, no tiene conexión con ningún servidor, significa que no actualiza su información y solo se puede navegar dentro de la misma página. Para crearla se utilizó HTML, CSS y JavaScript
- **En el escenario 2:** Vemos una arquitectura centralizada (todo está ubicado en el mismo sitio), contiene varias página web con contenido estática, a esto le llamamos sitio web estático, como en el caso anterior tampoco tiene conexión con otro servidor lo que significa que solo se puede navegar entre las mismas páginas. Para crearla se utilizó HTML, CSS y JavaScript.
- **En el escenario 3:** Vemos una Aplicación Web o Web Dinámica, en este caso además de utilizar HTML, CSS y JavaScript se utilizaron lenguajes de programación para poder hacer intercambio de información con las distintas capas de la aplicación, hacer cálculos, crear nueva información en la base de datos, actualizarla o conectar con otros sistemas mediante API.
- **En el escenario 4:** Vemos un Plataforma Digital o un Ecosistema, en este caso podemos ver que consiste en muchos sistemas que trabajan en conjunto, colaborando para resolver una necesidad o problema. En el ejemplo de Mercado Libre se dedica a vender online, pero también tiene que cobrar y hacer envíos. Por eso han desarrollado aplicaciones independientes pero que saben cómo comunicarse a otros sistemas para pedir o enviar datos para realizar alguna tarea. De esta manera las

aplicaciones pueden dar solución integral, comprar, pagar y enviar el producto sin tener que salir de la página.

Arquitectura web distribuida

Para comenzar tengamos presente que hemos visto que una aplicación web tiene varios conceptos como arquitectura cliente servidor, Front End, Back End, Base de datos y Apis. Pero qué pasa cuando una aplicación sabemos que va a recibir muchas consultas de nuestra cliente.

Para poder dar respuesta a ello y antes de entender que es una arquitectura distribuida hagamos el ejercicio de organizar cada concepto en su lugar y veamos cómo podemos distribuir nuestra aplicación.

Supongamos que nos juntamos varias compañeras del curso y tenemos un sistema web o aplicación web con la arquitectura Cliente / Servidor, en ella debemos organizar los conceptos que ya vimos como Front End, Back End, Base de datos y Api. Esta organización quedaría de esta manera:

Arquitectura Cliente / Servidor:

- **Cliente:**
  - Front End: El código o programación que muestra información al navegador web llamado cliente.
- **Servidor:**
  - Back End: El código o programación que ejecutas las acciones en el servidor y conecta con la base de datos.
  - Base de dato o DB: Donde se almacena la información.
  - Api: es quien nos permite conectar a otros sistemas.

¿Qué es una arquitectura centralizada?

Es cuando tenemos todos los elementos de nuestra aplicación web de arquitectura de Cliente / Servidor en un solo lugar equipo o servidor, es decir, tener el Back End, Front End, Bases de datos y APIs en el mismo equipo. Esto hace que en caso de una falla del equipo toda nuestra aplicación también fallará.

¿Qué es la arquitectura distribuida?

Es tener la posibilidad y capacidad de separar nuestro sistema en distintos servidores de la red (sea red local o internet). Ya sabemos que cuando hablamos de arquitecturas estamos refiriéndonos a una estrategia de cómo construir nuestro sistema dependiendo de lo grande que sea, de las funcionalidades que tenga, esto es más bien una forma de pensar en cómo escalar nuestro sistema para que soporte más usuarios o más transacciones.

¿Pero cómo te das cuenta cuando una aplicación es distribuida?

En este punto depende de que estés mirando el proyecto o la aplicación, te compartimos solo 2 enfoques:

- **Como usuario:** No te darías cuenta porque si el sistema está distribuido funciona como un conjunto único y sincronizado.
- **Como Programador:** Cuando te asignen a un proyecto o cliente en base a preguntas concretas podrás ir conociendo como se implementó o distribuyo el sistema, pero te compartimos algunas formas en las que puedes entender que estas frente a una arquitectura distribuida:
    -

    1. ***Por la Líder del Proyecto***: Cuando se comienza a trabajar en un proyecto generalmente la líder del mismo hace una explicación del tipo de aplicación con la que se está trabajando, además de indicarnos en qué parte del proyecto estaremos trabajando.
    2. ***Por el perfil asignado***: Cuando nos asignan el trabajo en una empresa nos especifican si trabajaremos en el Front End, en el Back End o en ambos Full Stack, de esa manera podemos inferir que la arquitectura es distribuida, igualmente siempre es mejor preguntar para estar seguros.
    3. ***Por un diagrama*****:** Generalmente se utilizan diagramas de aplicación para documentar un sistema, en el se puede ver la separación del sistema y si está distribuido en 1 o varios servidores.

En los siguientes diagramas o esquemas de una aplicación que fue creada y pensada en forma distribuida separando el código en Front End , Back End. Podemos ver cómo se puede ir escalando o distribuyendo en distintos servidores y que en cualquier caso seguirá funcionando.

ARQUITECTURA DISTRIBUIDA

1
NAVEGADOR CLIENTE
FRONT END
WEB SERVER
FRONT
BACK
BD SERVER

2
NAVEGADOR CLIENTE
FRONT END
HTTP
WEB SERVER
FRONT
BACK
TCP
BD SERVER

3
NAVEGADOR CLIENTE
FRONT END
HTTP
WEB SERVER
FRONT
HTTP
WEB SERVER
BACK
TCP
BD SERVER

4
NAVEGADOR CLIENTE
FRONT END
HTTP
Google Cloud
NUBE
WEB SERVER
FRONT
NUBE
Azure
HTTP
WEB SERVER
BACK
NUBE
Google Cloud
TCP
BD SERVER

Analicemos juntos los escenarios de la imagen anterior:

- **Escenario 1:** En este caso podemos ver que la aplicación está separada en Front End, Back End y bases de datos están en el mismo servidor. Notemos que la aplicación fue diseñada de forma modular o separada (para poder distribuirla) todas las partes del sistema están en un mismo servidor, es decir, en caso de falla del servidor afecta a todo el sistema.
- **Escenario 2:** En este caso podemos ver que se ha separado la base de datos y el sistema sigue funcionando porque el desarrollador Back End escribió el código pensando en una arquitectura distribuida. Pero la parte del Front End y Back End aún están en un mismo servidor.
- **Escenario 3:** En este caso podemos ver que cada parte del sistema Fornt End, Back End y Base de datos esta en un servidor diferente. Con esto comenzamos a ver los beneficios del diseño con arquitectura distribuida en los sistemas.
- **Escenario 4:** En este caso podemos ver que cada parte del sistema está en la nube de distintas empresas y nuestro sistema sigue funcionando por su diseño modular o distribuido.

¿Hasta dónde puedo modularizar o distribuir mi sistema?

El cómo distribuir el sistema es algo que se analiza en el diseño de la aplicación o se va cambiando a medida que el sistema va creciendo, normalmente cuando llegamos a un trabajo las aplicaciones ya están funcionando y necesitan de nuestro conocimiento para mantenerlas y agregarles mejoras.

Cuando una aplicación se hace más grande, compleja y con más usuarios necesitamos seguir modularizando el sistema, dado que no nos alcanza con separar en Front End, Back End y Base de datos. En esta situación debemos pensar en modularizar o separar algunas funcionalidades del sistema, algunos motivos pueden ser:

- **Por alta demanda**: Cuando el sistema tiene una funcionalidad que es compleja, consume mucho recurso del servidor o es muy demandada por distintas partes del sistema.
- **Por interconexión**: Cuando un sistema tiene funcionalidades que se necesita dar acceso a otros sistemas para consumir ese proceso, función o datos.
- **Por segregación de roles**: Cuando es necesario separar roles o funciones por motivos de seguridad o aspectos técnicos, también puede ser porque el negocio lo requiere, por ejemplo si se decide por seguridad separar el proceso de autenticación del sistema para reforzar la seguridad.
- **Por escalamiento:** Cuando las proyecciones indican que en un periodo de tiempo la demanda aumentara considerablemente, será necesario agregar más servidores en la red con la misma funcionalidad para que satisfaga la demanda.

Existen varias **formas de separar estas funcionalidades que llamaremos API REST o Microservicios**, si bien a medida que avancemos iremos aprendiendo más sobre las API REST, ahora veremos cómo la arquitectura distribuida puede aplicarse para pasar de un gran sistema que tiene todo en un solo lugar a separarlo en pequeñas y que todo siga funcionando.

Para ejemplificar tomaremos un sistema que tiene lo siguiente:

- **Fornt End:** Todos fue diseñado pensando en una arquitectura distribuida, las funcionalidades son las siguientes.
    - *Loguin:* Se conecta con el Back End para validar el usuario y clave.
    - *Cliente*: Se conecta con el Back End para consultar, editar, crear y eliminar clientes
    - *Producto*: Se conecta con el Back End para consultar, editar, crear y eliminar productos
    - *Proveedor*: Se conecta con el Back End para consultar, editar, crear y eliminar proveedores
- **Back End:** Todos fue diseñado pensando en una arquitectura distribuida, las funcionalidades son las siguientes,
    -
    - *Loguin*: Recibe la petición, consulta la base de datos y valida si el usuario.
    - *Cliente*: Recibe la petición, consulta la base de datos y devuelve los datos de cliente.
    - *Producto*: Recibe la petición, consulta la base de datos y devuelve los datos del producto.
    - *Proveedor*: Recibe la petición, consulta la base de datos y devuelve los datos del proveedor.

Ahora veamos en un diagrama o esquema cómo estas funcionalidades se pueden ir distribuyendo en distintos servidores:

ARQUITECTURA
MICRO SERVICIO API REST

1

FRONT END: LOGIN, CLIENTE, PRODUCTO, PROVEEDOR
BACK END: LOGIN, CLIENTE, PRODUCTO, PROVEEDOR
DB SERVER

2

FRONT END: LOGIN, CLIENTE, PROVEEDOR, PRODUCTO
BACK END: LOGIN
BACK END: CLIENTE, PROVEEDOR
BACK END: PRODUCTOS
DB SERVER

3

NUBE
Front End: LOGIN, CLIENTE, PROVEEDOR, PRODUCTO
NUBE
Back End: LOGIN
NUBE
Back End: CLIENTE, PROVEEDOR
NUBE
Back End: PRODUCTOS
NAVEGADOR CLIENTE
FRONT END
DB SERVER
MySqL

4

NAVEGADOR CLIENTE
FRONT END
NUBE
Front End: LOGIN, CLIENTE, PROVEEDOR, PRODUCTO
Back End: LOGIN, CLIENTE, PROVEEDOR, PRODUCTO
DB SERVER
MySqL

Analicemos juntos los escenarios de la imagen anterior:

- **Escenario 1:** En esta caso tenemos el sistema que está en 3 servidores para cada separación

1. Front End: Todas las funcionalidades están juntas y en el mismo servidor.
2. Back End: Todas las funcionalidades están juntas y en el mismo servidor.
3. Base de datos: Todos los datos en una base de datos y en el mismo servidor.

- **Escenario 2:** En este caso **podemos ver que cada parte del sistema está distribuido en 5 servidores** y el sistema sigue funcionando porque el desarrollador Back End escribió el código pensando en una arquitectura distribuida.

1. Font End: Todas las *funcionalidades están juntas* y en el mismo servidor.
2. Back End: La funcionalidad de *login está en un servidor* exclusivo.
3. Back End Las funcionalidades de *cliente y proveedor están en un servidor*.
4. Back End: La funcionalidad *producto está en un servidor* exclusivo.
5. Base de datos: Todos los datos en una base de datos y en el mismo servidor.

- **Escenario 3:** En este caso **podemos ver que cada parte del sistema está distribuido en 5 servidores en nubes diferente**s y el sistema sigue funcionando porque el desarrollador Back End escribió el código pensando en una arquitectura distribuida.
    1. Font End: Todas las *funcionalidades están juntas* y en el mismo servidor de la nube de Argentina.
    2. Back End: La funcionalidad de *login esta en un servidor de la nube* de España.
    3. Back End Las funcionalidades de *cliente y proveedor están en un servidor* de la nube de Canadá.
    4. Back End: La funcionalidad *producto está en un servidor en la nube* de Nueva Zelanda.
    5. Base de datos: Todos los datos en una base de datos y en un servidor propio.

- **Escenario 4:** En este caso **podemos ver que el sistema está en la misma nube,** a pesar que las funcionalidades están todas juntas como el en escenario 1 y nuestro sistema sigue funcionando.

1. Font End: Todas las *funcionalidades están juntas* y en el mismo servidor de la nube de Argentina.
2. Back End: Todas las *funcionalidades están juntas* y en la nube de Argentina.
3. Base de datos: Todos los datos en una base de datos y en la nube de Argentina.

Como hemos visto hay varios niveles de como modularizar, distribuir o separar en capas una aplicación con sus funcionalidades, en algunos cosos la separación solo puede ser Font End, Back End. Pero existen otras formas de separar en partes mas pequeñas la aplicación y eso lo hacemos con la ayuda de las APIs REST.

Tarea desafío a subir del módulo

Te recordamos e invitamos a que utilices este conocimiento para crear el esquema de arquitectura de tu Portfolio Web y lo subas en la sección: **Desafío a subir del módulo** (ver imagen de referencia).
El objetivo es poner en práctica todo lo aprendido. Para ello deberás considerar tanto el material de lectura como los contenidos desarrollados en los encuentros en vivo y los aportes de otros compañeros.

Evaluación diagnóstico >
Material de lectura >
Evaluación >
Resumen >
Master class >
Foro >
Desafío a subir del módulo >

Elementos de la arquitectura Web

A continuación, se enumeran los elementos de la arquitectura web (pueden variar según la arquitectura elegida):

1- La infraestructura de red
Si bien es cierto que, en fase de desarrollo, para probar nuestra aplicación web, no necesitaríamos esta infraestructura, una vez nuestra aplicación se instale en el hosting definitivo, será necesario una red ethernet y todos los componentes que hacen posible la conectividad de los equipos informáticos. Con esto nos referimos a cables de red, sea utp o fibra óptica, placas de red, sea wifi o cableada, switch, routers, modem, etc. Y estos componentes físicos son necesarios tanto del lado del cliente como del servidor. Con esto se quiere decir que no es posible implementar una aplicación web sino existe una infraestructura de red preexistente o se diseña e implementa una nueva. Esta puede ser: internet, intranet o extranet.

2.- Isp

Es el proveedor del servicio de internet.

3.- Cliente Web
Es el navegador web. Ejemplos: Chrome, Safari, Firefox, etc. Pero ya no se restringe solo a estos dispositivos sino que podría ser, por ejemplo, un sistema embebido ejecutándose en una SBC (small board computer). Incluso podría no estar ejecutando un navegador convencional, como por ejemplo un reloj inteligente, o un dispositivo vinculado a una máquina de producción seriada como los surgidos de la mano del concepto de Industria 4.0.

4.- Nombre de Dominio

Dicho de forma sencilla, el nombre de dominio (o simplemente "dominio") es el nombre de un sitio web. Es decir, es lo que aparece después de "@" en una dirección de correo electrónico o después de "www." en una dirección web. Si alguien te pregunta cómo te puede encontrar en Internet, tendrás que decirle tu nombre de dominio. Las computadoras para comunicarse utilizan direcciones IP (números únicos en la red). Un ejemplo de una dirección IP de un servidor es 173.194.121.32. Para nosotros es imposible recordar tantos números y saber qué servicio o qué aplicación se encuentrta en esa dirección IP o servidor. Para resolver estos problemas se usan palabras que las personas pueden leer, que son intuitivas, fáciles de recordar y dicen mucho sobre el servicio web que ofrecen, se denominan nombres de dominio. ¿Puedo comprar un nombre de dominio? No, los nombres no se pueden comprar, solo se puede pagar por el derecho a usarlo por cierto periodo de tiempo. Para registrar un dominio a tu nombre debes hacerlo por medio de una empresa que se encarga de administrar las registraciones de nombres de dominio. En el caso de Argentina es **https://nic.ar**
Aquí tienes algunos ejemplos de nombres de dominio: google.com, wikipedia.org, youtube.com https://domains.google/intl/es_es/learn/web-terms-101/

5.- URL

Una URL (o localizador uniforme de recursos) es una dirección web completa que se utiliza para encontrar una página web específica. Mientras que el dominio es el nombre del sitio web, la URL es una dirección que remite a los usuarios a una de las páginas de ese sitio web. Cada URL contiene un nombre de dominio y otros componentes necesarios para localizar una página o un contenido concreto. Aquí tienes algunos ejemplos de URL: http://www.google.com - https://es.wikipedia.org/wiki/umami - https://www.youtube.com/feed/trending ; https: // dominios. google / intl / es_es / learn / web-terms-101 /

6.- Sitio web

Aunque una cosa lleve a la otra, comprar un nombre de dominio no implica tener un sitio web. El dominio es el nombre del sitio web, la URL es la forma de

encontrarlo y el sitio web es lo que los usuarios ven en su pantalla y con lo que interactúan. Es decir, cuando compres un dominio, habrás adquirido el nombre de tu sitio web, pero te faltará crear el sitio web en cuestión. https://domains.google/intl/es_es/learn/web-terms-101/

7.- Servidor DNS

Sistema de Nombre de Dominio. Se ocupa de la administración del espacio de nombres de dominio. Este servidor se encarga de hacer las conversiones de nombres de dominio a direcciones IP. Cuando el cliente realiza una petición web, por ejemplo google.com, una de las primeras acciones del sistema es invocar un servidor DNS para que devuelva la dirección IP del / o de alguno de los servidores de google. Por ejemplo, devolverá la ip 172.217.162.14.

8.- Hosting

Es el nombre que se le da al servicio de alojamiento en la web a nuestras páginas, aplicaciones, bases de datos (los hosting son servidores que están siempre encendidos y conectados a internet). Los programadores, una vez terminado el trabajo suben su aplicación web al Hosting para que todo el mundo pueda acceder.

9.- Servidor Web

También llamado servidor HTTP, es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.

10.- Contenedor de aplicaciones Web (o servidor de aplicaciones web) ,

En el módulo que permite la ejecución de aplicaciones web. Por ejemplo, el módulo PHP o Python del Servidor Web. Componente ASP o ASPX de IIS. Servidor o Contenedor de Aplicaciones Web Java: Tomcat, Weblogic, Websphere, JBoss, Geronimo, etc.

11.- Servidor de Bases de Datos

Estos son contenedores de bases de datos que permiten organizar y administrar los datos que deben permanecer en un medio de almacenamiento permanente. Resuelven problemas de: seguridad, mecanismos de comunicación, concurrencia, inconsistencias de los datos, respaldo, entre otros. Hay varios tipos de bases de datos, por ejemplo, las relaciones que organizan los datos en forma de tablas, en filas y columnas. Otro tipo son los orientados a objetos u orientados a documentos donde el concepto de tablas se cambia por la colección con formatos similares a “json”. JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es

utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o viceversa).

¿Desarrollo Web full stack?

El *desarrollador web full stack* puede crear aplicaciones web **dinámicas**. Esto se logra en base a los **estándares web** y utilizando tecnologías web que pueden variar según la pila de desarrollo. Tiene por objetivo la creación de aplicaciones web dinámicas. Como vimos anteriormente, las aplicaciones web dinámicas modifican su contenido en función del usuario que acceden permitiendo mostrar uno u otro contenido dependiendo del usuario y de su interacción para con la aplicación web (ej. Facebook, Instagram, etc.) mientras que las aplicaciones web estáticas muestra siempre el mismo contenido independientemente del usuario visitante (ej. un blog).

Pero ¿qué son los estándares web y qué buscadores existen? Los estándares web son tecnologías que se utilizan para crear aplicaciones web. Los mismos son creados por organismos de estándares - instituciones que invitan a grupos de personas de diferentes compañías de tecnología a unirse y acordar cómo deben funcionar las tecnologías de la mejor manera posible para cumplir con todos sus casos de uso. El W3C es el organismo de estándares web más conocido, pero hay otros como WHATWG (responsables de la modernización del lenguaje HTML), ECMA (publica el estándar para ECMAScript, en el que se basa JavaScript), Khronos (publica tecnologías para gráficos 3D, como WebGL) y otras MDN Web Docs,

¿Pero qué son los Stacks?

Lo que denominamos ***stack tecnológico***, o también denominado ***stack de soluciones*** o ecosistema de ***datos***, es un conjunto de todas las herramientas tecnológicas utilizadas para construir y ejecutar una sola aplicación.

Recordemos lo que ya hemos visto en la arquitectura Cliente /Servidor a su vez al código que navega el cliente por el navegador le llamamos Front End y la parte de código que accede a los datos se le llama Back End , si no recuerdas has clic aquí,

Cada una de estas partes de la arquitectura puede crearse con distintos lenguajes de programación, el que quieras, por citar algunos ejemplos:

Front End: Para el desarrollo del Front End se necesita.

- Estructura y estilos: HTML, CSS o framework como Bootstrap
- Lenguaje programación: JavaScript, java, PHP o bien pueden ser framework como Angular

Back End: Para el desarrollo de Back End se necesita un web server, una base de datos y un lenguaje de programación.

- Base de datos: puede ser MySQL, PostgreSQL, etc
- Lenguaje: Java, PHP, etc
- Web server: Apache, NGINX, etc

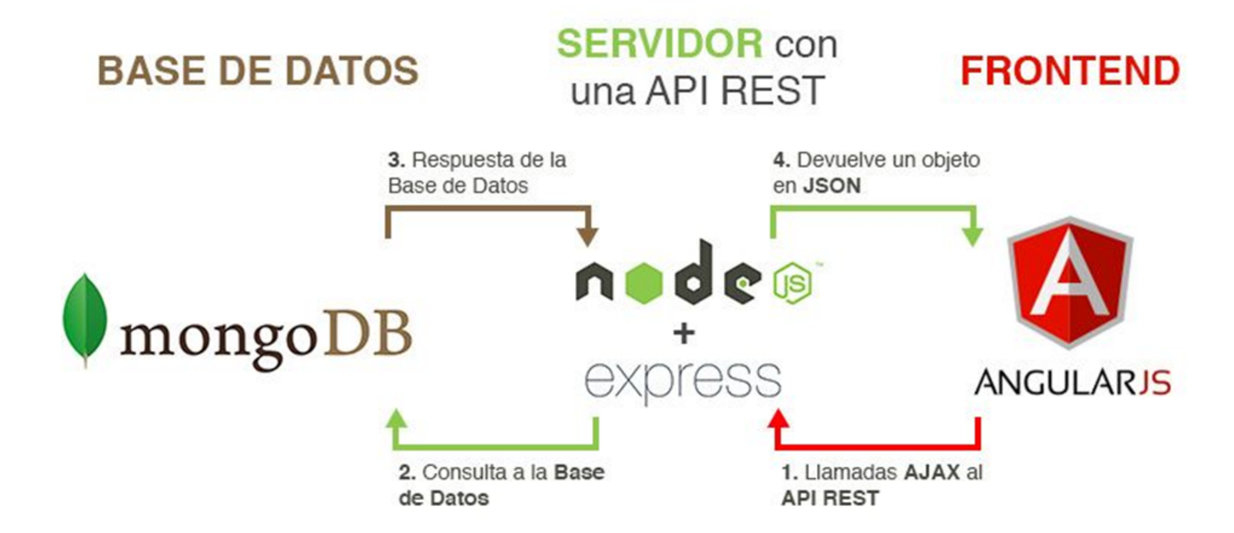
Como son muchas las opciones de lenguaje de programación, bases de datos y frameworks. Se ha creado el concepto de Stacks o pila que agrupa las tecnologías (que funcionan bien en conjunto o más utilizadas) para crear una aplicación desde el Front End y Back End usando ese Stack y ahorrarnos tener que investigar al detalle que tecnología podemos utilizar, es como un combo de tecnologías. Esto también ayuda a las empresas a identificar el conocimiento que tiene un programador al nombrar el Stack tecnológico que utiliza.

Sitios como la red social Facebook, han sido desarrolladas por una combinación de frameworks de codificación y lenguajes, entre los que se incluyen JavaScript, HTML, CSS, PHP y ReactJS. Así podemos decir que este es el "stack tecnológico" de Facebook.

Podríamos preguntarnos cuáles son los stacks que usan Netflix, Whatsapp, Instagram...

Uno de los stacks o pila de tecnologías más utilizado por los desarrolladores es el que se conoce por **LAMP** : **L**inux, **A**pache, **M**ySQL y **P**HP. Cualquier web hecha con Wordpress, Drupal o Prestashop, por ejemplo, están hechas sobre estos cuatro pilares. Pero se pueden hacer las variaciones que se crean convenientes, puesto que muchas de estas tecnologías son intercambiables por otras similares. Por ejemplo, NginX en lugar de Apache, PostgreSQL en lugar de MySQL o Ruby on Rails en lugar de PHP.

Otro stack muy utilizado es el llamado **MEAN** , que se compone de **M**ongoDB, **E**xpress, **A**ngular y **N**odeJS. te dejamos una imagen ilustrativa del Stack MEAN:



Backend para la Arquitectura de Aplicaciones Frontend. (s. f.). IBM Developer. Recuperado 11 de octubre de 2021, de https://developer.ibm.com/es/patterns/create-backend-for-frontend-application-architecture/

Fazt. (s. f.). YouTube. Recuperado 11 de octubre de 2021, de https://www.youtube.com/channel/UCX9NJ471o7Wie1DQe94RVIg

Algunos ejemplos de Stacks

- **LAMP**: Linux - Apache - MySQL - PHP. Es la más antigua y el formato de aplicaciones predominante se denomina ***"Multi Page Application o MPA"*** o aplicación de múltiples páginas.
- **MEAN**: MongoDB - Express - AngularJS - Node.js. Es la más moderna y el formato de aplicaciones predominante se denomina ***"Aplicación*** de una sola página o ***SPA"*** o aplicación de una sola página.
- **.NET:** .NET + WebApi + IIS - SQL Server. Utilizan Microsoft . Núcleo neto como plataforma de desarrollo.

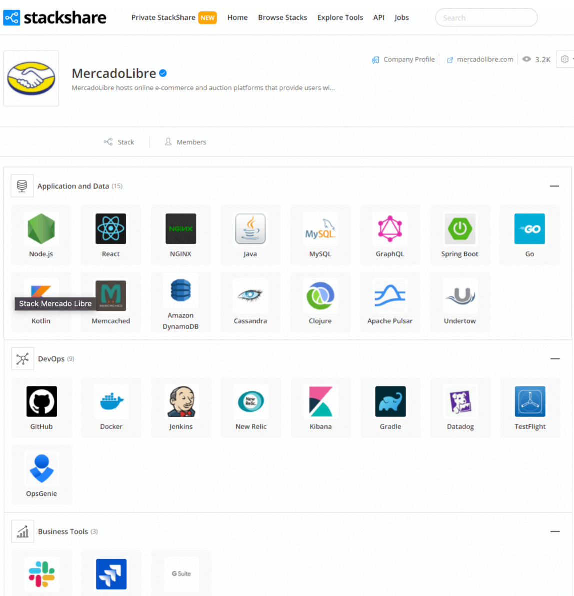
- **Patrón BFF**: Se puede usar una arquitectura Backend for Frontend (BFF) para crear backends para aplicaciones web o móviles orientados al cliente. Los BFF pueden ayudar a respaldar una aplicación con múltiples clientes al mismo tiempo que mueven el sistema a un estado menos acoplado que un sistema monolítico. El acoplamiento es "es el grado en que los módulos de un programa **depende** unos de otros" . Una aplicación monolítica es una unidad cohesiva de código. Este patrón de código ayuda a los equipos de desarrollo a iterar las funciones más rápido y tener control sobre los backends para aplicaciones móviles sin afectar la experiencia de la aplicación móvil o web correspondiente.
- **Django**: Python - Django - MySQL
- **Ruby on Rails**: Ruby - SQLite - Rails
- **LEMP**: Linux - Nginx - MySQL - PHP
- **MEER**: MongoDB - Express - React - Node.js.

- **Otros Stacks**: Exísten otros stacks y probablemente surjan nuevos en el futuro cercano. Una solución completa involucra no solo el stack, que en ocasiones conlleva la inversión en las correspondientes licencias, sino también las consideraciones referidas al costo de hardware, entre ellos el del servidor físico. Así por ejemplo, en torno al lenguaje de programación Java y de la mano del servidor web JBoss, se pueden implementar en servidores con sistemas operativos RedHat soluciones Web. El hardware puede ser por ejemplo de Dell, HP, entre otros. Otra posible configuración es el frontend desarrollado en torno a React, el backend en Python y Django y usando como base de datos SQLite

¿Todas las empresas usan si o si estos Stacks?

No siempre, tengamos en cuenta que los stacks son un grupo de herramientas tecnológica sugeridas para crear una aplicación web, por ende, las empresas pueden elegir las herramientas que necesiten sin tener en cuenta los Stack antes nombrados, Pero recuerda cuando una empresa solicita un programador LAMP y te presentas ellos asumirán que sabrás manejar ese grupo de tecnologías. Existen otros empresas que buscan perfiles con conocimiento solo en Front End o en Back End, estas empresas focalizan al programador en un solo área independientemente del stack.

En las siguientes imagenes te compartimos algunas de las tecnologías que utilizan unas empresas:



Software and technology stacks used by top companies. (s. f.). StackShare. Recuperado 11 de octubre de 2021, de https://stackshare.io/

¿Con que tecnología trabajaremos en el curso?

Luego de investigar y consultar la demanda requerida de las empresas vs las tecnologías más utilizadas en Argentina, utilizaremos las siguientes tecnologías:

Para Front End:

- Estructura y estilos: HTML, CSS y framework como Bootstrap
- Lenguaje programación: TypeScrypt y el framework como Angular ambos basados en Javascript

Para Back End:

- Base de datos: Usaremos MySQL
- Lenguaje: Java con Framework Spring Boot
- Web server: Apache Tomcat

Ahora que sabemos con las tecnologías que vamos a trabajar podemos decir que será ya tenemos nuestro stack tecnológico.

¿Qué significa ser un desarrollador full stack?

El desarrollador full stack es un perfil que tiene conocimiento de lenguajes de programación de Front End, Back End, APIs y Bases de datos. Esta amplitud de conocimiento es muy buscada hoy en día por las empresas, esto significa que al momento de trabajar en un proyecto puedes estar asignado a cualquiera de esas áreas.

Recuerda que tener el conocimiento de un programador full stack te amplia las posibilidades de trabajo, por ello, vale la pena el esfuerzo de aprender y practicar. En el proceso podrás descubrir que tienes facilidad o te gusta más alguna parte del Full Stack (Front End, Back End o Bases de datos) y posiblemente te conviertas en experto en alguno. Por eso a seguir estudiando y practicando.

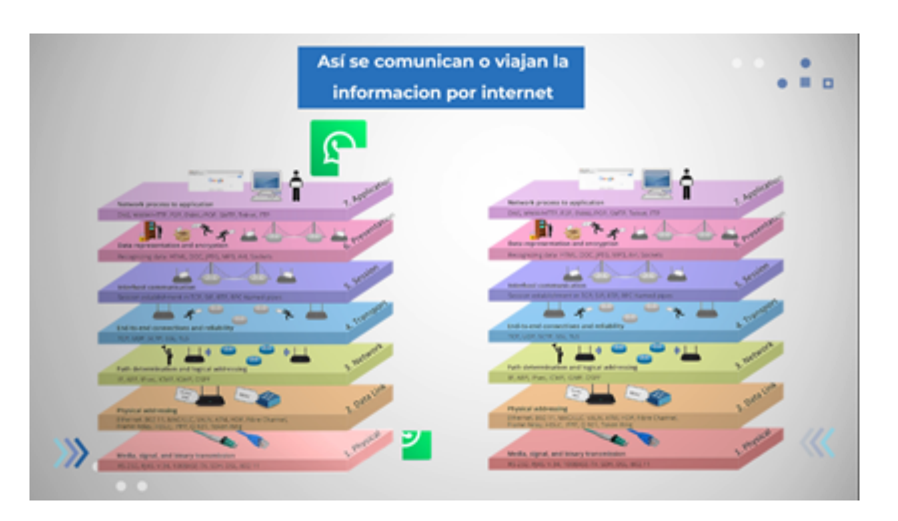
Proceso de una petición web- Modelo OSI

Ahora que entendemos los principios básicos de la arquitectura WEB y algunos de sus elementos, veamos el Modelo OSI.

El Modelo OSI o Modelo de Interconexión de Sistemas Abiertos (en inglés **O**pen **S**ystems **I**nterconection). Es un modelo de comunicación de 7 capas, y es la base por el cual viaja toda la información por las redes (internet global,

internet local, internet celular, etc.), te invitamos a profundizar más hacinado clic aquí

En la siguiente animación te mostramos como viaja la información por internet de una computadora a otra pasando por las 7 capas del modelo OSI.



Proceso de una petición web

1. Cliente Web: Solicita la resolución de nombres al servidor DNS. Por ejemplo: google.com
2. Servidor DNS: Recibe y trata la solicitud. Una vez recibida la petición realiza las consultas necesarias para resolver y obtener la dirección IP.
3. Servidor DNS: Devuelve al navegador Web la dirección IP que corresponde al Servidor Web.
4. Cliente Web: Conecta con el servidor web mediante la dirección IP y el puerto. Realiza la petición mediante una URL (Método GET) o un formulario (Método POST). Dicha solicitud incluye: la dirección IP del servidor web, el puerto del servidor web, URL y parámetros.
5. Servidor Web: Control de Acceso, Análisis de la petición y localización del recurso. Como detecta que es el acceso a un fichero o ruta de aplicación tiene que traspasar el control al Contenedor de aplicaciones Web
6. Paso de la petición del servidor web al contenedor de aplicaciones web
7. El contenedor analiza la petición y en base a la ruta traspasa el control a la aplicación web.
8. Paso del control de la petición desde el CAW a la aplicación.
9. La aplicación recibe la petición y decide qué hacer en base a ella, es decir, elegir la función que se encargará de gestionar esa petición, normalmente en base a la ruta, el método HTTP y los parámetros de entrada por URL. Una vez elegida ejecutará esa función.
10. La aplicación realiza una petición SQL a la base de datos.
11. La Base de Datos recibe la petición SQL y la procesa realizando los cambios que tenga que hacer, si corresponde.

12. Una vez procesada la petición devuelve los datos a la aplicación web, normalmente un conjunto de datos. Ej. los 10 últimos clientes.
13. La aplicación web recibe estos datos y tiene que generar una salida, normalmente HTML, donde estructura el contenido de los datos devueltos por la BBDD en etiquetas HTML.
14. La aplicación web devuelve una respuesta al Contenedor de Aplicaciones Web
15. El contenedor procesa la respuesta, para controlar la ejecución de la aplicación por si esta falla.
16. El Contenedor de Aplicaciones Web devuelve el fichero al servidor web.
17. El servidor Web devuelve los datos dentro de la respuesta HTTP al navegador web.
18. Cliente Web: Presenta (renderiza) el contenido HTML resultante.

Web 2.0

Back End

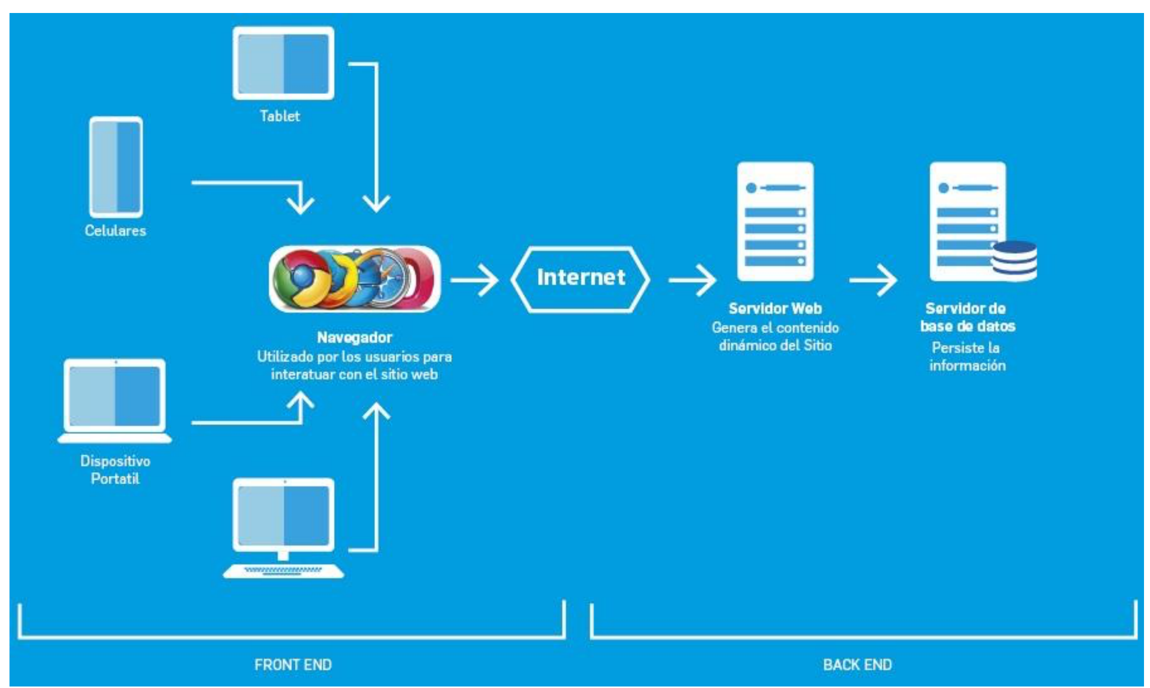
Consulta HTTP

18 Devolver Respuesta
4 Petición Web
Navegador Web
Servidor Web
5 Localizar Recurso
17 Devolver Respuesta
3 Devuelve IP
Consulta de DNS
1 Envío de Nombre
Devolver Respuesta 16
6 Paso de Control CAW
Servidor DNS
Procesar Respuesta 15
Contenedor Aplicaciones web
7 Localizar Aplicación
2 Búsqueda DNS
Devolver Respuesta 14
8 Paso de Control a la APP
Procesar Respuesta HTML 13
Aplicaciones web
9 Controlador Peticiones
Consulta BBDD 12
10 Consulta BBDD
Base de Datos
11 Procesado consulta

Arquitectura Web 2.0 (Dinámica en el servidor) Backend. (s. f.). Cursos de Desarrollo. Recuperado 11 de octubre de 2021, de
Proceso de petición web 2.0

En resumen, si hay algo que nos queda claro entonces, es el potencial que ha tenido y que tiene la web, su capacidad de escalabilidad, por lo que todos los mercados, incluso la cultura y el arte, se manifiestan con gran libertad.

Pero a medida que ampliamos los horizontes aparecen nuevos problemas y estos generan nuevas soluciones. Es que ya no son simples páginas web coloridas y dinamizadas, sino sistemas completos, distribuidos, multiplataformas, para usos generales y específicos, como por ejemplo una plataforma de e-commerce.



Arquitectura Web de 3 niveles

En cuanto a los desarrolladores, es importante agregar que si bien, lo más habitual es que se especialicen en frontend o en backend, hoy existe un tercer perfil: desarrollador “Full-Stack” (muy solicitado por las empresas de desarrollo de software). Este perfil se caracteriza por tener una visión integral de toda la aplicación web (frontend + backend).

Transferencia de datos

Conceptualización de las estructuras involucradas en la transferencia de datos entre computadoras

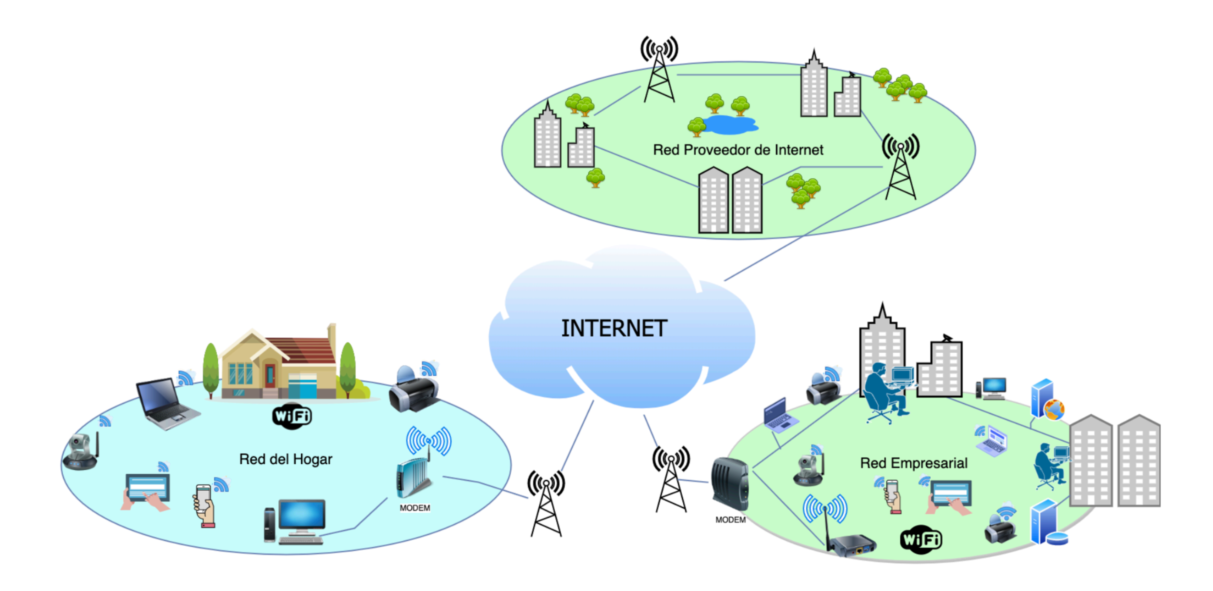
Para entender la programación web es necesario además conocer o tener alguna noción sobre las estructuras involucradas en la transferencia de datos entre computadoras. Esta transferencia de datos, es decir, la comunicación entre dos o más computadoras se lleva a cabo a través de lo que se llama normalmente red de computadoras. La más conocida red es Internet o, como se la denomina: la red de redes.

Vamos a ver conceptos de red de computadoras, de Internet y de cómo funcionan los elementos que las constituyen, a modo introductorio para entender cómo funcionan los sistemas para la Web.

Para comenzar vamos a definir que “ *(...) una red de computadoras, también llamada red de ordenadores, red de comunicación de datos o red informática, es un conjunto de equipos (nodos) y software conectados entre sí por medio de*

dispositivos físicos (cable) o inalámbricos (wifi) que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos con la finalidad de compartir información recursos y ofrecer servicios (...) " (Tanenbaum, 2003).

A continuación veremos un esquema de cómo se conectan distintas redes, como la red de un hogar con la red de una empresa y quien nos provee internet para transmitir o recibir datos.



Perez, C. J. (2021a, septiembre 12). Esquema de red interconectada [Imagen]. En *INTI*.

En la imagen anterior podemos observar los distintos escenarios:

- **Red del Hogar:** En la red del hogar podemos observar que entra el cable al modem wifi permitiendo que los equipos o nodos se conecten por la red inalámbrica, los dispositivos que se conectan son notebook, cámaras, celulares, impresoras. También podemos ver que desde el modem sale un cable hacia una computadora de escritorio. Todo está conectado por la misma red.
- **Red Empresarial:** En la red empresarial entra un cable al modem principal que llega al primer y segundo edificio permitiendo conectar en red a los equipos o nodos (notebook, pc de escritorio, impresora, servidores). También observamos que desde el modem principal sale un cable de red que conecta con un modem wifi el cual permite conectar los equipos o nodos que tenga conexión a la red por medio de wifi.

- **Red Empresarial:** En la red de proveedor de internet podemos observar que distintas empresas se conectan entre sí formando una red en la que transmiten y reciben datos.

¿Cual es el objetivo de una red?

El objetivo de las redes en general, consisten en "compartir recursos" entre computadoras de modo tal que los programas, datos y equipo estén disponibles para cualquier solicitud que los equipos de la red requieran, sin importar la ubicación o distancia física del recurso (equipo) y del usuario. Es decir, si tenemos una aplicación que necesita datos de otro equipo, sin importar que se encuentren a 1000 km de distancia de los datos, la aplicación podrá solicitar los datos, mostrarlos o utilizarlos como localmente.

La finalidad principal para la creación de una red de ordenadores es:

- ● Compartir recursos e información a grandes distancias (computadoras, impresoras, servidores, cámaras, etc.)
- ● Asegurar la confiabilidad y la disponibilidad de la información (características propias que definen lo que es información).
- ● Aumentar la velocidad de transmisión de los datos y obtenerlos en milisegundos para su uso.
- ● Reducir los tiempos y costos en actividades como procesamiento de información.

Si bien este no es un curso de redes de computadora es necesario conocer los elementos físicos en los que trabajan los sistemas que vamos a realizar. Con ello queremos decir que si bien podemos abstraernos de esta tecnología, es muy útil tener alguna noción de lo que pasa detrás de escena.

¿Como podemos entender una red?

La comunicación por medio de una red se lleva a cabo en dos categorías diferentes: una capa denominada física y otra lógica.

La **capa física** incluye todos los elementos de los que hace uso un equipo para comunicarse con otro equipo dentro de la red, como por ejemplo, tarjetas de red, los cables, las antenas, etc. Si te interesa conocer más podés leer autores como Tanenbaum en donde habla de redes y el modelo OSI (modelo de interconexión de sistemas, protocolos de comunicación, etc.) .

Con respecto a la **capa lógica** la comunicación se rige por normas muy rudimentarias que por sí mismas resultan de escasa utilidad. Sin embargo, haciendo uso de dichas normas es posible construir los protocolos denominados, que son normas de comunicación más complejas (de alto nivel) capaces de proporcionar servicios útiles.

Los protocolos son un concepto muy similar al de los idiomas de las personas. Es decir, si dos personas hablan el mismo idioma, y respetan ciertas reglas (tales como hablar y escucharse por turnos), es posible comunicarse y transmitir ideas e información. Ese es el modelo de un protocolo.

Para formar una red se requieren elementos de hardware, software y protocolos. Los elementos físicos se clasifican en dos grupos: los dispositivos de usuario final (llamados también Hosts) y los dispositivos de red. Entre los dispositivos de usuario final podemos enumerar computadoras, impresoras, escáneres, y demás elementos que brindan servicios directamente al usuario. Los segundos (dispositivos de red) son todos aquellos que conectan entre sí a los dispositivos de usuario finales posibilitando su intercomunicación. (Farías, 2013).

Internet es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP / IP, lo cual garantiza que las redes físicas heterogéneas que la componen, constituyan una red lógica única de alcance mundial. (Guillamón, 2009)

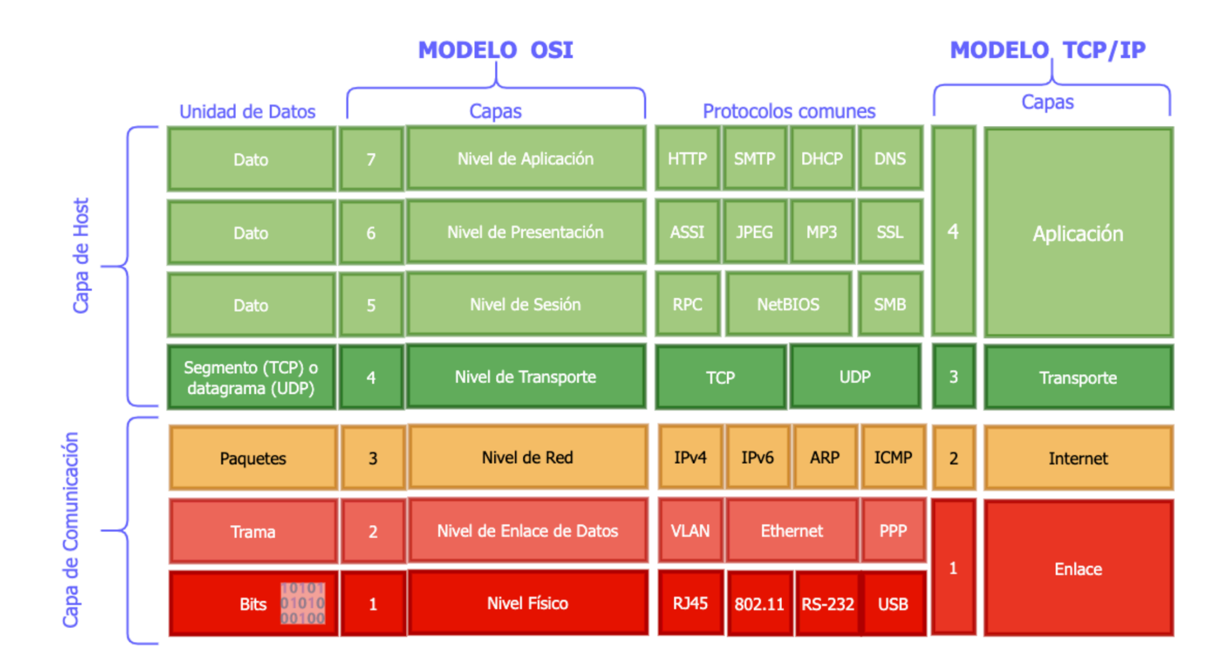
Uno de los servicios que más éxito ha tenido en Internet ha sido world wide web, WWW o la web. La WWW es un conjunto de protocolos que permite de forma sencilla la consulta remota de archivos de hipertexto y utiliza Internet como medio de transmisión («Internet, n.». Oxford English Dictionary (Draft edición). Marzo de 2009).

Para hacer más notable esta distinción vamos a nombrar algunos otros servicios y protocolos en Internet aparte de la web por ejemplo para el envío de correo electrónico usamos el protocolo SMTP, para la transmisión de archivos usamos el protocolo FTP, para nombrar algunos.

Esta familia de protocolos es un conjunto de protocolos de red en los que se basa Internet y permite la transmisión de datos entre computadoras como hemos dicho.

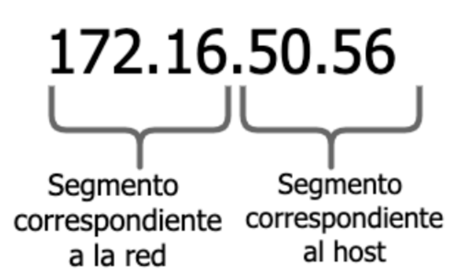
Entre los principales podemos nombrar el conjunto de protocolos TCP / IP que hace referencia a los dos protocolos más importantes que componen la Internet, que fueron los primeros en definirse y qué son los más utilizados.

TCP (protocolo de control de transmisión) se usa para crear conexiones entre computadoras a través de las cuales pueden enviarse un flujo de datos. Por la forma en la que está implementado este protocolo los datos serán entregados en su destino sin errores y en el mismo orden en el que se transmitieron. ¿Esto qué quiere decir? que es un protocolo orientado a conexión, ya que el cliente y el servidor deben anunciarse y aceptar la conexión antes de comenzar a transmitir los datos entre ellos. Es decir que hay un intercambio de mensajes entre ellos para abrir una línea de conexión que permanece abierta durante toda la comunicación. (dsi.uclm.es 2007)



Fuente: [Oracle (3/10/2020) Parte II Administración de TCP / IP](#)

Por otro lado, el protocolo IP es un protocolo cuya función principal es el uso direccional en origen o destino de comunicación para transmitir datos mediante un protocolo no orientado a conexión que transfiere paquetes conmutados a través de distintas redes previamente enlazadas según la norma OSI.



Fuente: [Oracle (3/10/2020) Parte II Administración de TCP / IP](#).

Algo importante del diseño del protocolo IP es que se realizó suponiendo que la entrega de los paquetes de datos sería no confiable por eso se tratará de realizar del mejor modo posible mediante técnicas de enrutamiento sin garantías de alcanzar el destino final, pero tratando de buscar la mejor ruta entre las conocidas por la máquina que está usando IP.

Para entender mejor esta distinción entre dos protocolos uno TCP y otro IP habría que analizar el modelo de capas OSIen éste curso no profundizaremos sobre esto, pero lo que sí vamos a decir es que hay una jerarquía entre capas y el protocolo IP

pertenece a una capa denominada de red que está por encima de una capa denominada de transporte en dónde se encuentra TCP.

Entonces, en conclusión, se utiliza la combinación de estos dos protocolos para la comunicación en Internet, en donde TCP aporta la fiabilidad entre la comunicación e IP la comunicación entre distintas computadoras ya que las cabeceras de IP (cabecera por ser una parte el protocolo) contienen las direcciones de destino de las máquinas de origen y llamadas direcciones IP. Estas direcciones serán usadas por los routers para decidir el tramo de red por el que se enviarán los paquetes.

Para entender mejor el funcionamiento de la Internet vamos a decir que **dentro de la red de redes que es Internet debe existir un mecanismo para conectar dos computadoras. Este mecanismo lo proporciona el protocolo de Internet, el cual hace que un paquete de una computadora, llegue a la otra de manera segura a través del protocolo TCP y que llegue a destino a través de las direcciones IP.**

Para terminar, una dirección IP es un número que identifica de manera lógica y jerárquica una interfaz de un dispositivo dentro de una red que utiliza el protocolo de internet. Todas las computadoras en internet tienen una dirección IP, en la imágen siguiente vemos un esquema de 2 computadoras con sus direcciones IP.



Perez, C. J. (2021a, septiembre 12). Esquema de red interconectada [Imagen]. En *INTI*.

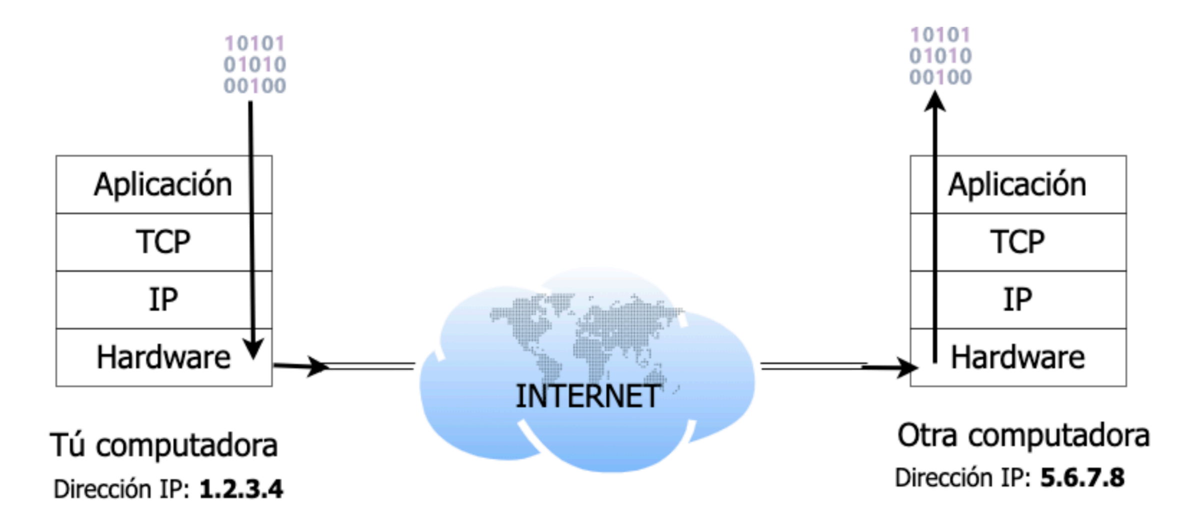
A modo informativo vamos a decir que existe otro sistema que se denomina sistema de nombres de dominio o DNS que asocian nombres comunes a direcciones IP por ejemplo la dirección www.cba.gov.ar tiene asociado un número de IP correspondiente, pero este mecanismo existe para que sea más fácil llegar a esa página web sin tener que recordar el número de IP.

Con esto concluimos una introducción de lo que es la comunicación entre computadoras en internet. Si bien parece algo complejo es de mucha importancia conocer estos mecanismos con los que trabajamos.

Por ejemplo, si queremos comunicarnos con una base de datos en Internet y nos pide un número de IP ya sabemos de qué se trata. También sabemos que

podemos comunicarnos a cualquier computadora conectada en red a través de mecanismos y que nos abstraemos de cómo se hace esto. Es decir, no nos tenemos que preocupar de manejar errores o interferencias en la comunicación.

Ya vimos que la estructura de red se maneja en capas. También mencionamos que hay una capa de red en dónde está el protocolo IP, una capa Superior de transporte en dónde está el protocolo TCP y ahora vemos una nueva capa que es la de aplicación en dónde se usa el protocolo HTTP.



Fuente: Oracle (3/10/2020) Parte II Administración de TCP / IP.

Por otro lado, antes de continuar con el desarrollo web tradicional tenemos que aprender un nuevo protocolo que es imprescindible para la comunicación en la web.

HyperText Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) (**Protocolo de Transferencia de Hipertexto** en español**)** es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque puede ser utilizado para otros propósitos también.
HTTP define un conjunto de **métodos de petición** para indicar la acción que desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados *verbos HTTP*.

Es muy importante saber que HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. Luego veremos las implicancias de esto.

Una descripción importante del protocolo es que **está orientado a transacciones y sigue el esquema de petición / respuesta entre un cliente y un servidor**. El cliente realiza una petición enviando un mensaje con cierto formato al servidor. El

servidor le envía un mensaje de respuesta. Para hacerlo más concreto, un cliente podría ser un navegador web y un servidor podría ser una aplicación en un servidor web corriendo en Internet.

Para que entendamos la forma de cómo se programa una aplicación web, necesitamos en alguna medida entender los mensajes HTTP. Los mensajes HTTP son en texto plano lo que hace más legible y fácil de depurar. Estos tienen la siguiente estructura:
.- Primero, hay una línea inicial en donde se diferencian dependiendo de si son peticiones y respuestas. Para las solicitudes la línea comienza con una acción requerida por el servidor, a esto se le denomina método de petición seguido de la url del recurso y la versión http que soporte al cliente. Lo importante es el método de petición y la URL (Uniform Resource Locator o localizador de recursos uniforme).

.- Para las respuestas, la línea comienza con la versión de HTTP seguido por un código de respuesta y con una frase asociada a dicho retorno.

.- También los mensajes tienen una cabecera que son metadatos con información diversa y el cuerpo de mensaje que es opcional. Típicamente este cuerpo tiene los datos que se intercambian entre el cliente y el servidor.

Los métodos de petición (o también llamados verbos) son varios. Cada método indica la acción que desea que se efectúe sobre el recurso identificado lo que este recurso representa dependiente de la aplicación del servidor.

Los métodos que vamos a usar son los de **get** y **post** . Cabe destacar que hay convenciones en donde se utilizan otros métodos. En este punto entran en juego el estándar REST y las API's.

El método get solicita una representación del recurso especificado. Las solicitudes que usan sólo deben recuperar datos y no deben tener ningún otro efecto.

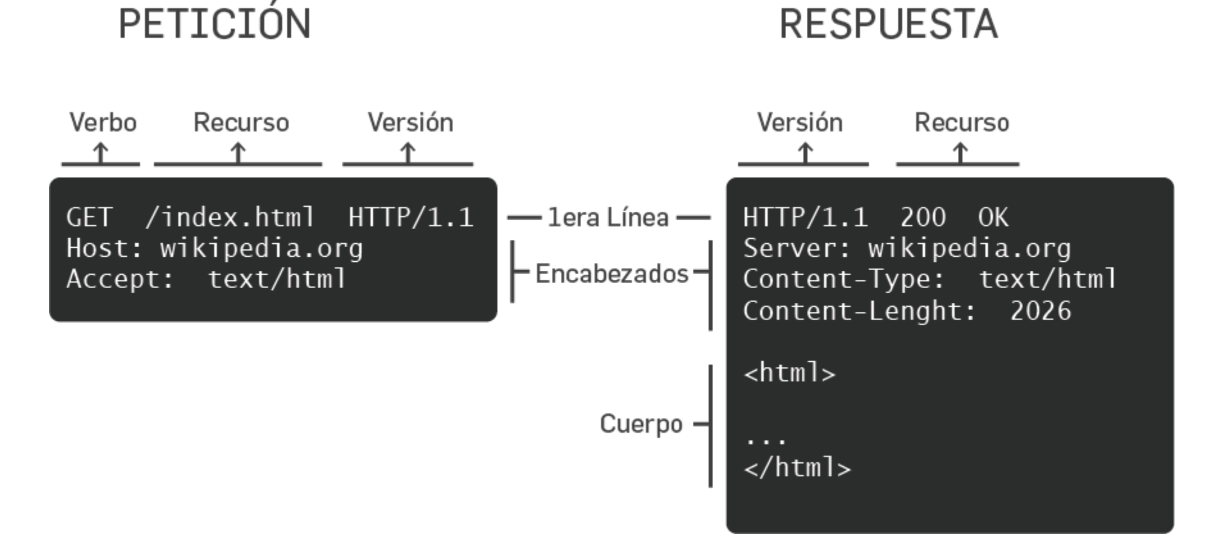
El método post envía los datos para que sean procesados por el recurso identificado. Los datos enviados se incluirán en el de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes.

Para finalizar con la explicación de HTTP vamos a nombrar códigos de respuesta:

1. Por ejemplo, el 200 representa una respuesta correcta es decir que indica que la petición ha sido procesada correctamente.
2. Otro ejemplo es la respuesta 404 que significa errores causados por el cliente, por ejemplo, que el recurso no se encuentra.
3. Finalmente, las respuestas que comienzan con 5 por ejemplo el 500 son errores causados por el servidor. Esto indica que ha habido un error en el proceso de la petición a causa de un fallo en el servidor.

Todo lo que hablamos de HTTP lo vamos a usar al momento de programar una aplicación web. Vamos a ver que al presionar por ejemplo un enlace se hace una petición “get” al servidor para buscar otra página lo que resultará en una respuesta 200 si se encuentra la página o en un 404 si no se encuentra. También veremos que al llenar un formulario y enviarlo al servidor lo haremos a través de una petición post.

Los otros protocolos de TCP / IP son necesarios para configuraciones de conexiones, pero a nivel de aplicación no los utilizaremos. Eso no significa que no haya que entenderlos pues nos facilitarán la solución de problemas, por ejemplo, al conectarnos a una base de datos desde la aplicación web del servidor.



[Mensaje http]

Method	Description	Sec.
GET	Transfer a current representation of the target resource.	4.3.1
HEAD	Same as GET, but only transfer the status line and header section.	4.3.2
POST	Perform resource-specific processing on the request payload.	4.3.3
PUT	Replace all current representations of the target resource with the request payload.	4.3.4
DELETE	Remove all current representations of the target resource.	4.3.5
CONNECT	Establish a tunnel to the server identified by the target resource.	4.3.6
OPTIONS	Describe the communication options for the target resource.	4.3.7
TRACE	Perform a message loop-back test along the path to the target resource.	4.3.8

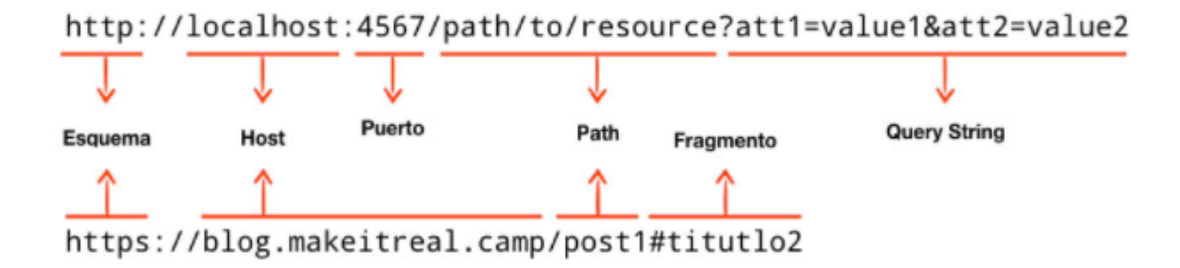
[Verbos y cómo se relacionan con operaciones]

Código de estados

Cuando el servidor devuelve una respuesta se indica un código de estado:

- **1xx**: Mensaje informativo.
- **2xx**: Exito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- **3xx**: Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- **4xx**: Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- **5xx**: Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable

Código de respuesta HTTP:



URL: Composición de la URL

Glosario

Bases de datos*:* Una base de datos es una colección organizada de información estructurada, o datos, típicamente almacenados electrónicamente en un sistema de computadora (https://www.oracle.com/mx/database/what-is-database)

DOM: El *DOM* **(Modelo de Objetos de Documento** en español **Modelo de Objetos del Documento)** es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse

de un elemento, una cadena de texto o un comentario).
(https://developer.mozilla.org/es/docs/Glossary/DOM)

Frameworks: La traducción literal de *framework* es ' **marco de referencia** ', y explica muy bien lo que significa. Un *framework* es un patrón o esquema que ayuda a la programación a estructurar el código ya ahorrar tiempo y esfuerzos a los programadores. (https://fp.uoc.fje.edu/blog/que-es-un-framework-en-programacion/)

LAN: *LAN* (red de área local) describe una red cuya área geográfica no se extiende más de una milla
(http://www.frlp.utn.edu.ar/materias/info2/Redes%20Lan.html)

Python: Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad
(https://desarrolloweb.com/articulos/1325.php)

Rails: Ruby on Rails (o simplemente Rails) es un framework Open Source, multiplataforma y basado en el lenguaje de scripting Ruby que facilita sobremanera el diseño y desarrollo de aplicaciones web que acceden a bases de datos. Rails separa automáticamente en tres capas todos los componentes de una aplicación web (Model, View y Controller: MVC), lo que hace más sencillo y rápido el mantenimiento de aplicaciones que en otros entornos
(https://www.spri.eus/euskadinnova /es/enpresa-digitala/agenda/introduccion-ruby-rails-completo/709.aspx#:~:text=Ruby%20on%20Rails%20)

Swift: Swift es un lenguaje intuitivo de programación de código abierto creado por Apple que permite diseñar aplicaciones para iOS, Mac, el Apple TV y el Apple Watch. (https://www.apple.com/es/swift/).

WAN: (Wide Area Network) describe una red cuya área geográfica excede a la de una ciudad. A menudo, varias LAN o varias MAN son enlazadas para crear una WAN (http://www.frlp.utn.edu.ar/materias/info2/Redes%20Lan.html)

Introducción gestión del tiempo

Introducción

"El tiempo es un recurso escaso, limitado y finito por ello se debe gestionar. Se utiliza o se pierde, es tu elección y decisión que hacer con él"

Comencemos con una situación hipotética ¿Qué harías con tu tiempo hoy, si fuera tu último día de vida? Para responder esto, no es necesario saber qué es el

tiempo, pero si es importante saber cómo lo administramos. Podemos tomar como medida un día. Cada día tiene 24 horas que son 144 minutos o 86.400 segundos.

El tiempo, en función del objetivo al que esté destinado puede dividirse en:

1. **Tiempo Personal:** Este es el tiempo que decidimos dedicarlo a cualquier actividad de nuestra elección personal haciendo uso de nuestra libertad de elección y de nuestro tiempo libre de compromisos.
2. **Tiempo Laboral:** Este es el tiempo que decidimos intercambiarlo con alguien que nos paga por hacer algún trabajo.
3. **Tiempo Familiar y Social:** Este es el tiempo que decidimos dedicar a las actividades sociales como fiestas, visitas a amigos, familia, etc.

Como el tiempo es finito y se necesita repartir en muchos ámbitos o actividades es necesario organizarse, priorizar y tener pequeñas metas



El tiempo como recurso limitado

Revisemos algunas cosas:

Uno de los problemas de los/as CEOs de empresas y cualquier profesional de cualquier actividad, es tener la sensación de que el tiempo del que disponemos no es suficiente para hacer todo lo que se nos presenta. Sin embargo, cada uno de nosotros **disponemos de este mismo *recurso, escaso y limitado,*** y siempre nos parece insuficiente. Además, sabemos que el tiempo es imprescindible para cualquier aspiración, deseo o actividad humana.

Existen algunas actividades que pueden ser obligatorias o necesarias en nuestras vidas como estudiar, trabajar, descansar, divertirse, comer, etc.… todo esto exige tiempo. Cuando se tiene alguna aspiración (un sueño o un deseo) acaba siendo cuestión de tiempo si logras organizarte.

Te proponemos que pienses y te respondas las siguientes preguntas:

- ¿Tenes que estudiar y no tuviste tiempo?
- ¿Trabajas al máximo, pero igual no llegas?
- ¿Trabajas a full por eso no podés ver a tu familia?
- ¿Trabajas un montón y por eso no podés hacer deportes?
- ¿Tenes mucho trabajo y por eso no podés leer "ese libro"?
- ¿Los amigos se juntan, pero no podes ir porque estas muy ocupado trabajando?

Aristóteles citaba «Somos lo que hacemos día a día. De modo que la excelencia no es un acto, sino un hábito»

Pero entonces, ¿por qué ocurre esa sensación constante de no tener tiempo y no saber qué hacer para remediarlo? Cuando del tiempo se trata, se dicen muchas cosas, pero la mayoría no tiene una confirmación o verificación concreta, ¿sabías que la ciencia aun no puede determinar qué es el tiempo con exactitud? También existen otras ramas que investigan todo esto, pero sin ser tan técnicos te vamos a poner 2 posibles motivos por el cual pudiéramos tener una deficiente gestión del tiempo, pero solo ahondaremos en uno de ellos.

1. **Déficit cognitivo**: Este punto se refiere al funcionamiento de nuestro cerebro, es decir, el déficit es que nuestras habilidades cognitivas no son suficientes para la gestión que intentamos realizar. Un ejemplo podría ser la falta de concentración, la incapacidad para organizarse, etc. Estas funciones ejecutivas, pueden ser la habilidad de planificar el tiempo, organizar y priorizar información, revisar la ejecución, etc. que no importa si tenemos un déficit, ya que existen ***estrategias compensatorias*** para cada dificultad incorporándolas como hábitos y así poder tomar las riendas sin problemas.

2. **Hábitos o rutinas incorrectos**: Este punto es el más importante y debemos mirarlo con más detenimiento porque depende enteramente de nosotros observarlo, plantear una estrategia y trabajar para cambiarlo. Los hábitos o rutinas pueden ser Mentales o Físicas, pero en cualquier caso siempre es posible enfocarlas hacia objetivos planificados y a medida que va transcurriendo el tiempo se manifiestan en resultados concretos.

Compartinos como administras tu tiempo completando con estas breves prguntas haceindo clic aquí

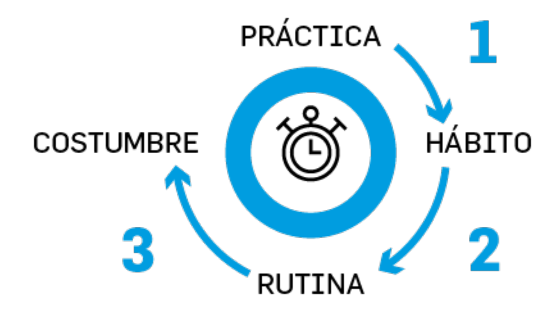
Ciclos y hábitos en el tiempo

Relación entre la programación (Ciclos) y los hábitos

Hemos mencionado los hábitos o rutinas, y la programación nos brinda algunas analogías para poder entender mejor estos términos. En la programación de

sistemas aprendemos el por qué y para qué sirven el **bucle** o **ciclo** (es una sentencia que se ejecuta hasta cumplir la condición), es decir es una forma de gestionar el tiempo dentro del programa o enfocarlo a hacer algo específico hasta cierto tiempo o condición.

Analicemos la relación entre un bucle de programación y un bucle de la vida real para generar un hábito. Un hábito se genera cuando hacemos algo muchas veces, luego que ya lo repetimos muchas veces y no nos cuesta decir que se ha generado un hábito.

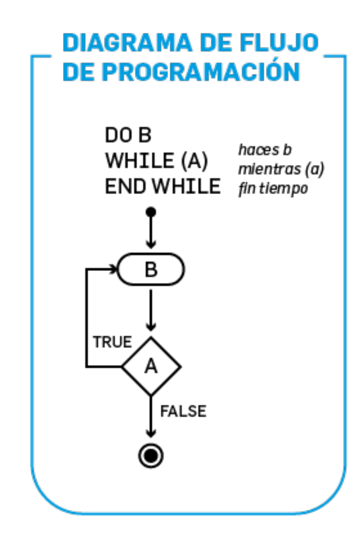


Tomemos la imagen circular de arriba que nos muestra el **ciclo de la práctica hacia la costumbre**.

El recorrido es:

1. De la **Práctica** se pasa al **Hábito**,
2. luego del **Hábito** pasa a la **Rutina,**
3. por último de la **Rutina** pasa a la **Costumbre**.

Ahora repasemos lo que es un ciclo en programación y usémoslo en la práctica a ver cómo queda usando diagramas y pseudocódigo.





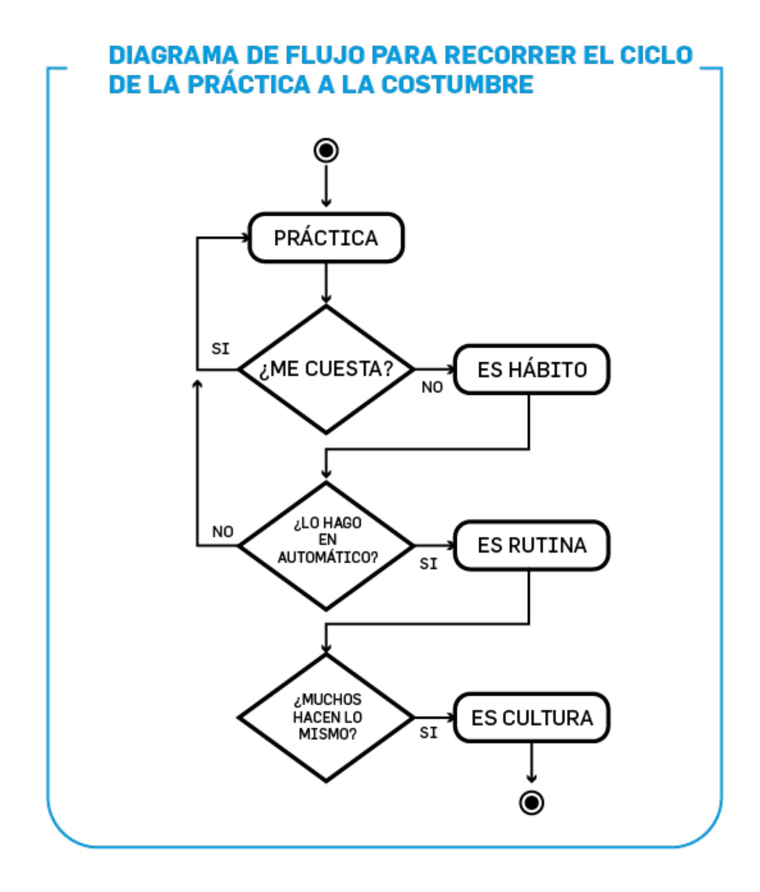




2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein



Completemos un poco más el diagrama hasta llegar al hábito para ver cómo queda:



¡Recuerda! que este es un ejemplo ilustrativo de cómo la programación nos ayuda a modelar o pensar conceptos abstractos aplicando una lógica y unos procesos ordenados para resolver problemas o llegar a un resultado.

Entonces, reformulemos las preguntas importantes: Si tienes un mal hábito ¿Cuánto debes hacer para que generes un buen hábito? o ¿Cuánto tiempo crees que te llevaría cambiarlo? y ¿Qué malos hábitos crees que tienes?

¡Pensemos cómo resolverlo!…

1. ¿Necesitas más tiempo?
2. ¿Necesitas 2 relojes como el Diego?
3. ¿Estas estresado?

Tal vez lo que necesitas de verdad es decidir mejor y elegir en qué invertís tú tiempo. ***Recuerda que tener tiempo = tener oportunidad de hacer***

Análisis Gestión del tiempo

Ahora veamos, ¿Cómo gastas tu tiempo hoy? Hagamos una cuenta rápida: usemos 2 actividades (dormir y comer) para ver cuánto tiempo nos sobra:

ACTIVIDADES	HORAS
Dormir	8
Comer	2

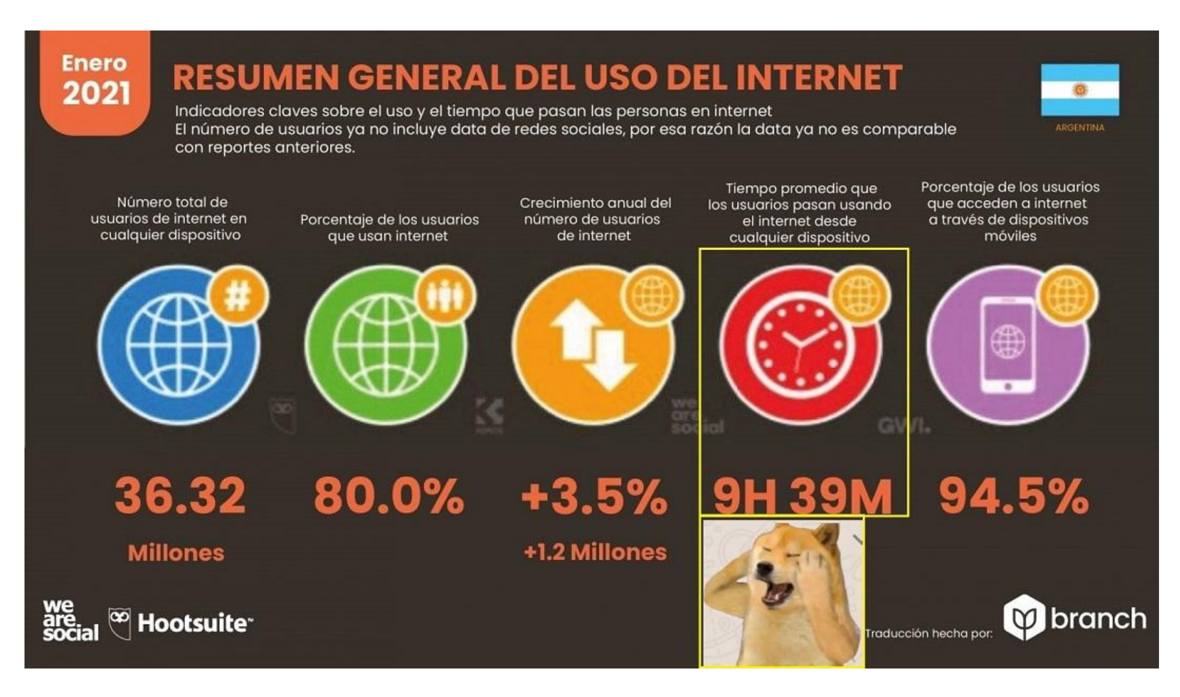
10 ⟶ Sobran horas

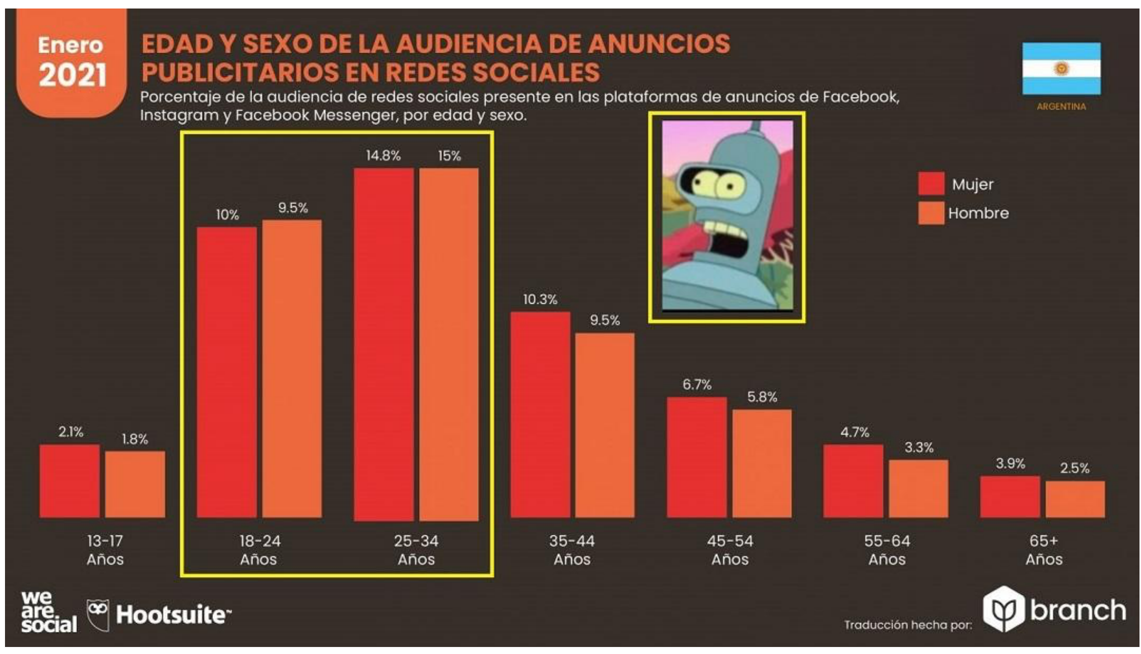
\- 24 horas tiene el día

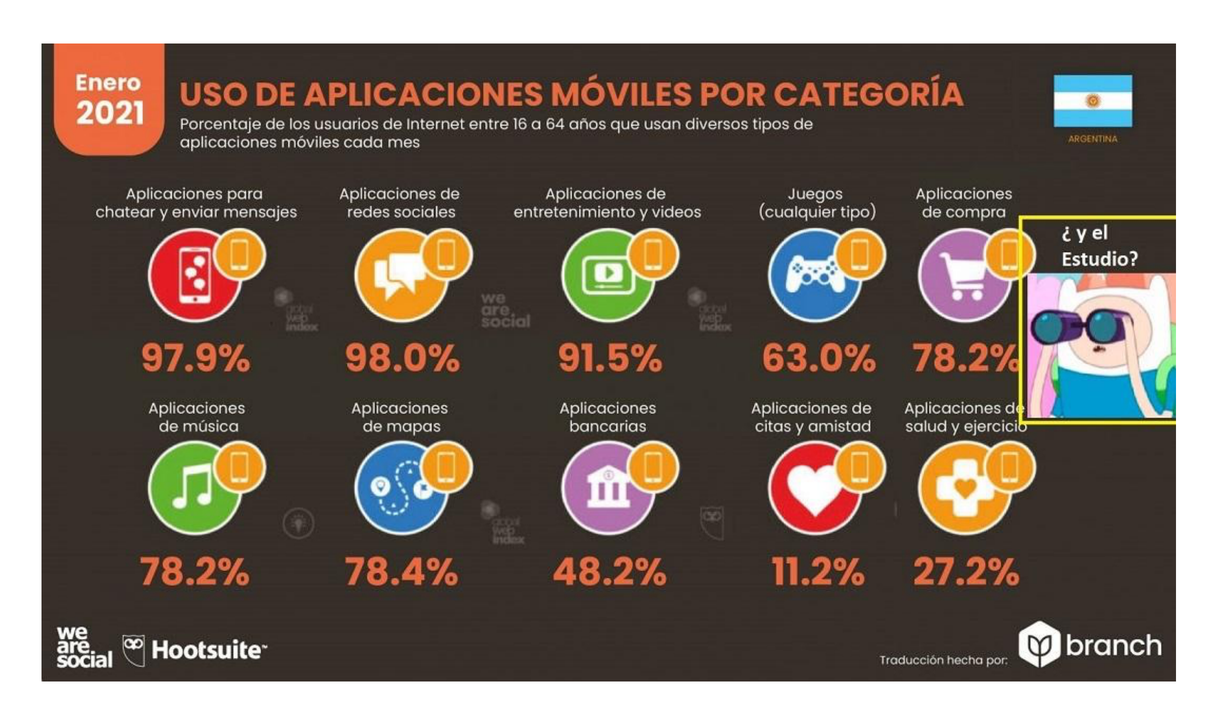
Sobran 14 horas del día

Indicadores Sociales de consumo del tiempo

Te mostramos cómo la Argentina utiliza internet y en qué lo utiliza para que trates de ver cómo consumimos nuestro tiempo.


Enero
2021
RESUMEN GENERAL DEL USO DEL INTERNET
Indicadores claves sobre el uso y el tiempo que pasan las personas en internet
El número de usuarios ya no incluye data de redes sociales, por esa razón la data ya no es comparable con reportes anteriores.
ARGENTINA
Número total de usuarios de internet en cualquier dispositivo
Porcentaje de los usuarios que usan internet
Crecimiento anual del número de usuarios de internet
Tiempo promedio que los usuarios pasan usando el internet desde cualquier dispositivo
Porcentaje de los usuarios que acceden a internet a través de dispositivos móviles
36.32
Millones
80.0%
+3.5%
+1.2 Millones
9H 39M
94.5%
we are social
Hootsuite
Traducción hecha por:
branch


Enero
2021
EDAD Y SEXO DE LA AUDIENCIA DE ANUNCIOS PUBLICITARIOS EN REDES SOCIALES
Porcentaje de la audiencia de redes sociales presente en las plataformas de anuncios de Facebook, Instagram y Facebook Messenger, por edad y sexo.
ARGENTINA
Mujer
Hombre
2.1%
1.8%
13-17 Años
10%
9.5%
18-24 Años
14.8%
15%
25-34 Años
10.3%
9.5%
35-44 Años
6.7%
5.8%
45-54 Años
4.7%
3.3%
55-64 Años
3.9%
2.5%
65+ Años
we are social
Hootsuite
Traducción hecha por:
branch





Metodología de Organización del tiempo (GTD)

¿Qué opciones tenemos para organizarnos? ¿Existen metodologías? Las metodologías pueden ser varias, pero siempre debes investigar para ver cuál se adapta mejor a tus hábitos y actividades. Por el momento te presentamos 2 metodologías (GTD y TimeBoxing) y algunas formas de cómo podés aplicarlo a tu calendario.

GTD

¿Qué es GTD? Getting Things Done (GTD) es un método de gestión de las actividades y el título de un libro de David Allen que en español se ha titulado "Organízate con eficacia". GTD se basa en el principio de que una persona necesita liberar su mente de las tareas pendientes guardándolas en un lugar específico. De este modo, no es necesario recordar lo que hay que hacer y se puede concentrar en realizar las tareas. Siempre investiga más del tema arrancando por acá .

FLUJO DE TAREA
Y ACCIÓN

1 RECOPILAR TAREAS

Revisión por día o Semana
- Procesar tus notas
- Revisar tus citas
- Revisar citas futuras
- Revisar tu lista de proyectos
- Lista de próximas acciones
- Lista de control
- Lista algún día
- Revisar material de apoyo

2 PROCESAR

LISTA DE TAREAS
Tarea 1
Tarea
Tarea
.
.
.

3 ORGANIZAR

4 REVISAR

LISTA DE PROYECTOS
-Some quick
-Example text to build on the card
-Title amd make up the bulk oh
the card´s content

TAREAS / ACCIONES

PLANIFICAR Tareas / Acciones

#	Task	Start	Effort
1	task1	20/10/2022	40h
2	task2	20/10/2022	40h
3	task3	20/10/2022	40h

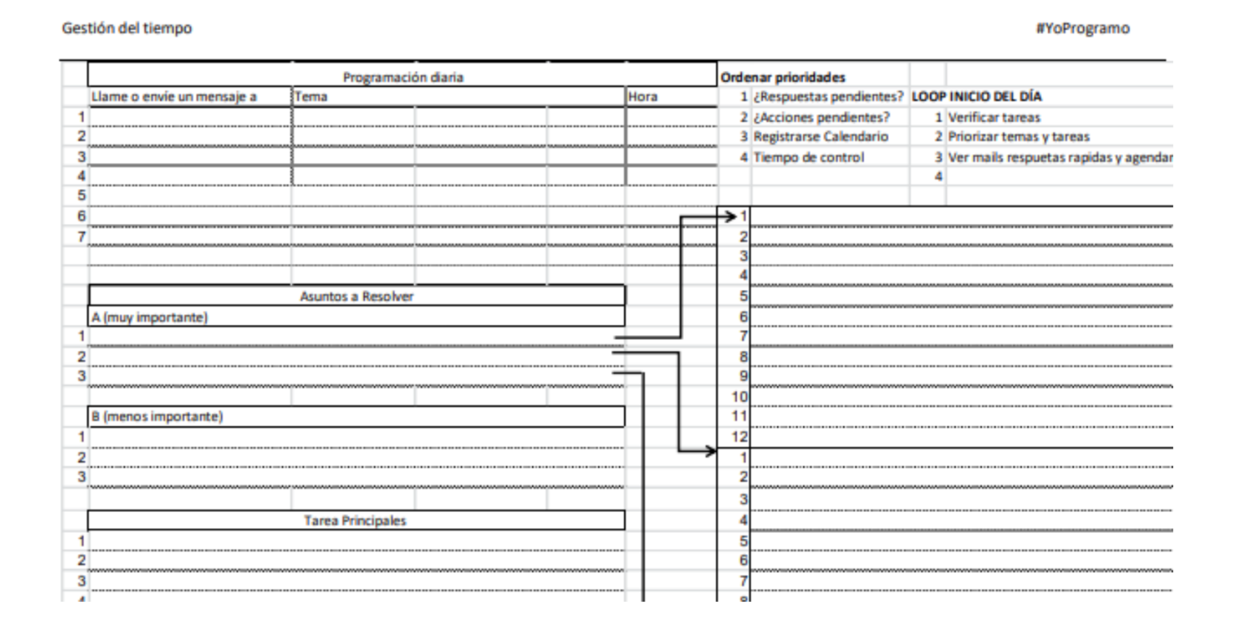
¿PUEDO HACER ALGO?
NO → ELIMINAR / INCUBAR / ARCHIVAR
SI

¿TIENE FECHA?
NO → ESTABLECER FECHA CONCRETA
1.- Prioridad
2.-Fecha
3.- Horas dedicadas
SI

5 EJECUTAR ACCIONES PLANEADAS

TAREAS / ACCIONES

Te dejamos un documento PDF en el cual puedes registrar tus tareas, prioridades y llevar a la acción las actividades. Descarga el archivo haciendo clic aquí.


Gestión del tiempo #YoProgramo

Programación diaria
Llame o envíe un mensaje a | Tema | Hora
1 2 3 4 5 6 7

Asuntos a Resolver
A (muy importante)
1 2 3
B (menos importante)
1 2 3

Tarea Principales
1 2 3

Ordenar prioridades
1 ¿Respuestas pendientes?
2 ¿Acciones pendientes?
3 Registrarse Calendario
4 Tiempo de control

LOOP INICIO DEL DÍA
1 Verificar tareas
2 Priorizar temas y tareas
3 Ver mails respuetas rapidas y agendar
4

1 2 3 4 5 6 7 8 9 10 11 12
1 2 3 4 5 6 7

Time Boxing

¿Qué es el Timeboxing? Timeboxing es una estrategia de gestión del tiempo orientada a los objetivos que puede ayudarte a aumentar la productividad y evitar la procrastinación. Cuando fijas un “bloque de tiempo”, estableces una meta para finalizar una tarea específica dentro de un período determinado. Planificar con anticipación cuánto tiempo te llevará finalizar una tarea te permitirá organizar de manera consciente cómo usarás tu tiempo y en qué trabajarás. Esto puede ayudar a reducir los pormenores del trabajo, que actualmente ocupan el 60% de nuestro tiempo . En lugar de perseguir aprobaciones o buscar documentos, puedes asegurarte de tener al alcance todo lo que necesitas al momento de empezar a trabajar en tu bloque de tiempo. Podes empezar por acá .

Si quieres implementar la técnica de timeboxing para mejorar tu concentración o combatir la procrastinación, prueba estos siete consejos para gestionar el tiempo de manera eficaz:

1. Prioriza los bloques de tiempo establecidos

Cuando tienes un día muy ocupado, puede resultar tentador reorganizar, reprogramar o cancelar los bloques de tiempo reservados para ese día. Trata de hacer todo lo posible para evitar eso. Una vez que hayas creado bloques de

tiempo, considéralos como reuniones programadas contigo mismo. No se trata de una planificación vaga. Por el contrario, se trata de un compromiso que genera contigo mismo para trabajar en esta tarea durante el tiempo que has reservado. Del mismo modo que no cancelarías una reunión con un miembro del equipo a último momento a menos que sea absolutamente necesario, tampoco debes cancelar un bloque de tiempo.

2. Visualiza tu tiempo para comprenderlo mejor

El timeboxing es más efectivo cuando lo puedes visualizar. Programar el tiempo para la concentración en tu calendario te ayuda a visualizar qué límite de tiempo tienes para cada tarea en particular. También te ayuda a cumplir con tus horarios y les permite a los miembros del equipo saber cuándo no estarás disponible. Si no marcas estos tiempos para la concentración en tu calendario, tus compañeros de equipo pueden interpretarlo como tiempo libre y programar una reunión o enviarte consultas.

3. Configura un temporizador

Independientemente de si estableces un bloque de tiempo rígido o flexible, usa un temporizador para programar una alarma que te notifique cuándo se acaba el tiempo. Recuerda: cuando creas un bloque de tiempo, lo que haces es establecer un objetivo para finalizar una tarea en particular en un período determinado. El temporizador puede ayudarte a cumplir con esa expectativa y notificarte cuando se acabe ese tiempo.

El temporizador también te permite trabajar sin tener que pensar en la hora. En lugar de consultar el reloj constantemente o tener que preocuparte por no llegar tarde a tu próxima reunión, puedes usar ese tiempo para sumergirte de lleno y centrarte en tu trabajo.

En el caso de los bloques de tiempo rígidos, prográmate pasar a la siguiente tarea una vez que se acabe el tiempo. Si aún no has finalizado esa tarea, puedes retomarla en un bloque de tiempo posterior. En los bloques de tiempo flexibles, usa el temporizador como un recordatorio de que debes concluir lo que estás haciendo en los próximos cinco o diez minutos y pasar a la siguiente tarea.

4. Desactiva las notificaciones o programa el modo “No molestar”

Los bloques de tiempo son efectivos ya que te ayudan a concentrarte en la tarea en cuestión y sumergirte de lleno en el trabajo. A menudo desviamos nuestra atención o la misma se ve interrumpida por cuestiones externas, lo que además de provocar que nuestra productividad disminuya, también impide que la realicemos de un modo eficaz y de alto impacto. Del mismo modo, tampoco querrás distraerte innecesariamente una vez iniciado tu bloque de tiempo.

Si desactivas las notificaciones o activas el modo “No molestar”, puedes asegurarte de que no te interrumpirán y, al mismo tiempo, informa a los demás miembros del equipo que te comunicarás con ellos lo antes posible. La mayoría de las herramientas permiten que los miembros del equipo "ignoren" estas configuraciones en caso de que sea necesario ponerse en contacto contigo, por lo que no estarás completamente aislado, simplemente estarás protegido de notificaciones innecesarias mientras trabajas en tu bloque de tiempo.

5. Tómate descansos entre bloques de tiempo

Cuando hablamos de productividad, no nos referimos a “hacer todo lo que puedas”. Y ese tampoco es el objetivo del timeboxing. Por el contrario, creemos que las estrategias de gestión del tiempo son una forma de maximizar tu eficiencia y lograr el mayor impacto posible en tu trabajo.

De un modo similar, use el método timeboxing para organizar tu trabajo no es una forma de lograr la mayor productividad posible, sino una forma de ayudarte a tu tiempo de forma más consciente y eficaz. Como todo trabajo que requiere mucha concentración, asegúrate de tomar descansos breves entre bloques de tiempo, aunque solo sea para ponerte de pie y estirarte o beber un poco de agua.

6. Captura todo tu trabajo en una herramienta de gestión del trabajo
Para crear un bloque de tiempo, primero necesitas saber en qué tienes que trabajar. Si aún no lo has hecho, crea una lista de tareas pendientes en un proyecto o en una herramienta de gestión del trabajo como Google, Trello o Asana. Estas herramientas te ayudan a identificar qué tareas importantes tienes por delante y qué tareas debes realizar durante cada día de trabajo.

7. Agrupa las tareas similares

Para implementar el timeboxing de manera eficaz, debes crear bloques de tiempo individuales para la mayoría de las tareas o iniciativas. Sin embargo, aun con todos los beneficios del timeboxing, cambiar entre tareas requiere tiempo y energía mental. Para evitar esto, recomendamos agrupar tareas similares en bloques de tiempo consecutivos. Esto permite que tu cerebro se mantenga en la misma "sintonía", incluso cuando estés trabajando en iniciativas individuales. Agrupar tareas similares puede ayudar a mantener la concentración y el flujo, incluso al pasar de un bloque a otro.

Por ejemplo, si formas parte del equipo de ventas, seguramente necesites preparar presentaciones y redactar correos electrónicos para diversas oportunidades de ventas. Lo ideal es definir bloques de tiempo para cada tarea. Pero en **este** caso, intenta agrupar las tareas relacionadas con el mismo cliente en bloques de tiempo consecutivos. Por ejemplo, si necesitas preparar una presentación y redactar un email para la empresa A, y preparar una presentación para la empresa B, fija las tareas para la empresa A en bloques de tiempo consecutivos antes de ponerte a trabajar en la presentación para la empresa B.

Hagamos un ejemplo:

Imagina que necesitas estudiar y no sabes por dónde arrancar, si es así, responde estas preguntas:

1. **Pensemos qué tienes que hacer:**

- Definí cuál es el objetivo y cuál sería la consecuencia.

2. **¿Qué tiempo disponible tienes en el día?:**

- Fíjate en la semana qué días y qué horas tienes libre para eso que necesitas hacer.

3. **¿Cuántas cosas debes hacer en ese tiempo?:**

- Arma una lista de tareas a realizar y métela s dentro de ese espacio de tiempo que tienes disponibles.

4. **¿Cuándo debes hacer cada cosa?:**

- Es momento de abrir la agenda y crear el TimeBoxing de 2 horas, asignar las tareas que crees que puedes hacer en ese tiempo.

5. **¿Te alcanzará el tiempo asignado?:**

- Es común que calculemos mal el tiempo al principio, así que, si no te alcanzo *no pasa nada* , tenes que asignar las tareas que te faltaron de la lista de tareas para el próximo TimeBoxing

Si respondes esas preguntas seguramente vas a saber por dónde empezar, ahora respondámosla:

1. **Pensemos qué tienes que hacer:**

- Tengo que estudiar antes que me atrase y no entienda nada.

2. **¿Qué tiempo disponible tengo en el día?:**

- Los lunes, miércoles y viernes tengo 2 horas libres.

3. **¿Cuántas cosas debo hacer en ese tiempo?:**

- Tengo que *(hacer resumen, investigar del tema, leer 3 webs y practicar)*

4. **¿Cuándo hago cada cosa?:**

- Bloqueo en la agenda 2 horas que necesito para el *lunes (hago el resumen y leer), miércoles (la investigación) y viernes (practicar)*.

5. **¿Me alcanzó el tiempo asignado ?:**

- Evalúo si me alcanzo el tiempo y si me falto crear un nuevo timeboxing para otro día.

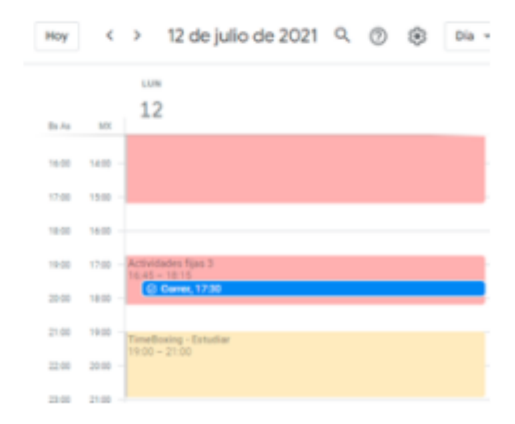
Organizar Agenda

Organización - Agenda

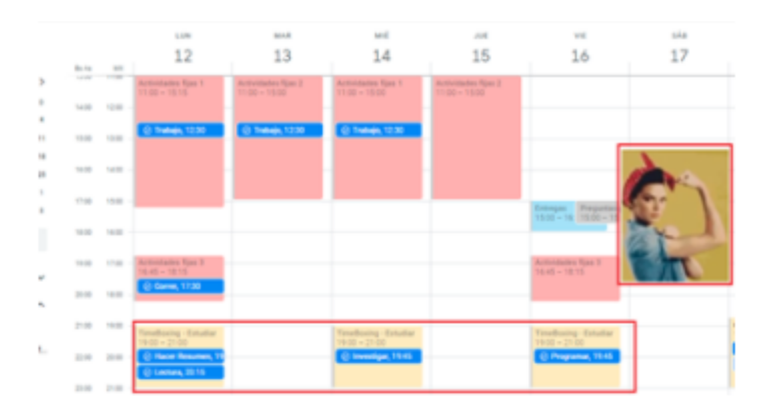
¿Cómo se ve todo lo que planteamos con la agenda? Te mostramos como quedaría todo organizado...



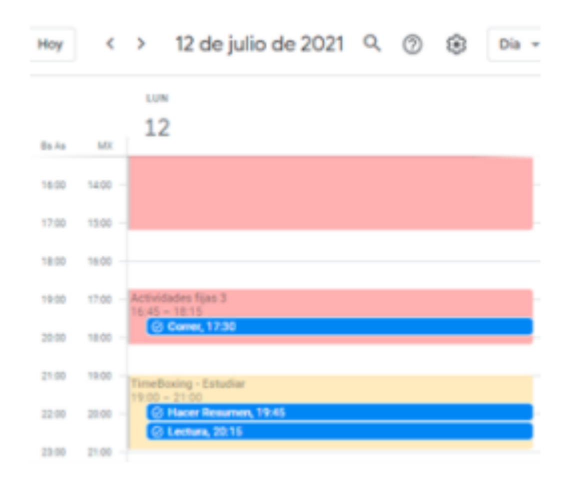
Timebox asignado - Vista por semana



Timebox asignado - Vista por día



Tareas asignadas - Vista por semana



Tareas asignadas - Vista por día

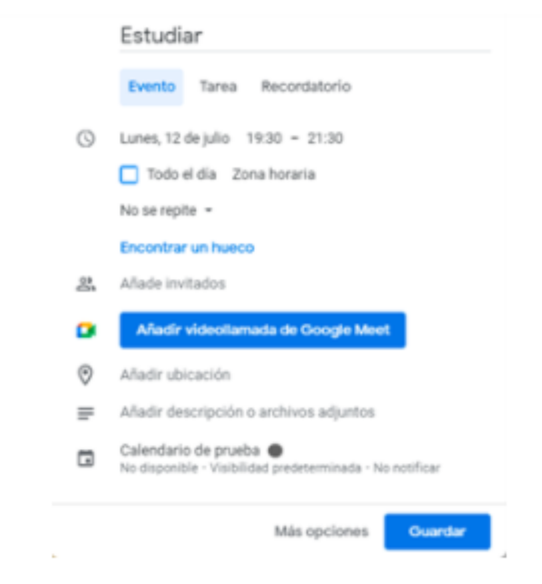
En este punto te podemos recomendar que utilices la vista por semana para organizar tus TimeBoxing (bloques de tiempo) ya que te permite ver de manera general los tiempos generales de la semana.



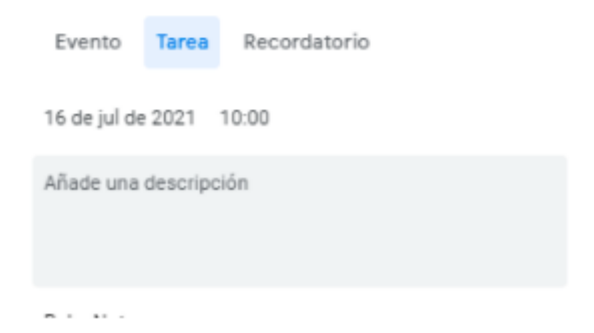




2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein



Calendario de Google - Crear timebox



Calendario de Google - Crear tarea

Planifica tu tiempo disponible

Para finalizar te dejaremos un archivo de Excel en el cual podrás calcular tus horas libres y así planificar tu agenda, puedes descargarlo haciendo clic aquí. ten presente que puedes crear tu propio Excel.

1. Define cuanto horas tienes libre

	AM									
Horas de tu día:	6	7	8	9	10	11	12	13	14	15
Marca con una x tus horas libres		X		X					X	

(*) Marca con una x las horas que tengas libre

Tus horas libres en el día son: 3

2. Planifica tu tiempo para estudiar

(*) Modifica los valores según necesites o corresponda

	Dale Prioridad	1	2	3	4	5	6	7	8	9
	Nombre actividad	Modulo 1	Modulo 2	Modulo 3	Modulo 4	Modulo 5	Laravel	Spring boot	Java	AWS + Spring
	¿Cuantas hs tarda el curso?	13	13	6	2.5	3	16	13	11	6
	Cuantas hs vas a estudiar x día?	1.5	1	1	1	1	2	2	2	1
	Vas a tardar estos días en terminar	8.7	13	6	2.5	3	8	6.5	5.5	6
	Te falta planear estas hs	2	7	0	0	-1	6	13	11	6
Día y horas libres	**Horas de estudio**									
Semana 1 *Lunes 3hs libres*	1	1								
Martes 3hs libres	0									
Miercoles 3hs libres	1					1				
Jueves 3hs libres	1	1								
Viernes 3hs libres	1	1								
Sabados 3hs libres	0									
Domingo 3hs libres	2	2								
Semana 2 *Lunes 3hs libres*	2	1	1							
Martes 3hs libres	3.5	2	1.5							
Miercoles 3hs libres	2.5	1	1.5							
Jueves 3hs libres	3	1	1	1						
Viernes 3hs libres	4	1	1	1	1					

Perez, C. J. (2021, 15 marzo). Planilla excel planificadora de tiempo [Imagen]. En *INTI*.

Tarea desafío a subir del módulo

Te recordamos e invitamos a que utilices este conocimiento para crear tu agenda de estudio y del proyecto así podrás subirlo en la sección: **Desafío a subir del módulo** (ver imagen de referencia).
El objetivo es poner en práctica todo lo aprendido. Para ello deberás considerar tanto el material de lectura como los contenidos desarrollados en los encuentros en vivo y los aportes de otros compañeros.

Evaluación diagnóstico ❯
Material de lectura ❯
Evaluación ❯
Resumen ❯
Master class ❯
Foro ❯
Desafío a subir del módulo ❯

¿Cómo gasto mi tiempo?

¿CÓMO GASTO ACTUALMENTE MI TIEMPO?

Con la realización del primer ejercicio has podido darte cuenta de la idea que tienes de cómo gastas el tiempo actualmente. Una forma de conocer mejor en qué lo gastas es realizar un diario de tus actividades, tareas, tiempo de ocio, etc., utilizando un registro de las 24 horas del día definiendo bloques de actividades.

1. ¿Cuánto tiempo tienes realmente?
2. ¿Qué eventos fijos tienes? *(Por ejemplo, Actividad física - Salud)*
3. ¿Qué tarea recurrente o repetitiva tienes? *(Planificación / Gestión - Ejecución - Entregas - Reuniones - formación)*
4. ¿Cuál es la prioridad?
5. Ponerlo en la agenda

Utiliza la información anterior, del horario confeccionado, para estimar cómo distribuyes actualmente tu tiempo, y cómo sería tu distribución ideal. Marca los sectores en función del porcentaje por áreas:

- A: Vida Personal (hobbies, tiempo de descanso, relajación… para ti mismo/a)
- B: Familia directa, pareja…
- C: Vida Social (reuniones de amigos, salidas de fin de semana…)
- D: Trabajo (incluye aquí trabajo remunerado y tareas domésticas)
- E: Estudio

El tiempo y Yo - MIS NECESIDADES

¿A qué quieres o necesitas dar tiempo en tu vida en este momento?.

METAS:

Establecer metas puede ser algo motivador, ayuda a centrar las ideas. Pero las metas, ya sean a corto o largo plazo, no son algo rígido. A veces es necesario modificar las metas propuestas, y no significa que hayamos fallado o que seamos más débiles que el resto de las personas. Es sólo una respuesta realista a una situación concreta. Una meta es un sueño con una fecha límite, o al menos eso se dice. Por ejemplo:

- Quiero ser maestro.
- Quiero estudiar magisterio.
- Quiero estudiar 2º curso de magisterio.
- Quiero aprobar 5 asignaturas este cuatrimestre.
- Quiero estudiar esta semana una asignatura.
- Hoy voy a estudiar un tema de una asignatura.

Tener sueños, y desear alcanzarlos, pueden ayudarte a tener momentos muy productivos, generando y floreciendo buenas ideas.

Eficiencia vs eficacia

¿Cómo nos damos cuenta de que tenemos una mala organización del propio trabajo y un mal uso del tiempo?

Una buena forma de saber si tienes una mala organización del trabajo y mal uso del tiempo es a través de los siguientes síntomas:

1. Jornadas cada vez más largas
2. Cuando te piden algo, No saber o no querer decir "no"
3. Buscas el perfeccionismo y se te va el tiempo en eso
4. No negocias plazos de respuesta
5. Empleas mal el tiempo de entrevistas (escuchar, preguntar, argumentar)
6. Tienes objetivos personales confusos o cambiantes
7. Falta de un plan de trabajo diario
8. No dejas un tiempo para imprevistos
9. Utilizas mal las comunicaciones (Internet, chats, series, películas, consolas de juego, etc.)
10. Escapar a hacer el compromiso de la tarea
11. No quieres tener ningún objetivo
12. Desconocimiento de las personas que influyen en una decisión
13. Abordar las tareas en tiempo real, hacerlo deprisa, uno mismo
14. Estrés o cansancio permanente

¿Vos qué crees, quién es el culpable de que en tu caso el tiempo no te rinde?

Cuando revisemos juntos cómo utilizas tu tiempo, te vas a sorprender al descubrir que ***son tus propios hábitos*** los que te permiten o no un aprovechamiento adecuado de este recurso finito.

¡¡¡Acá viene el momento esperanzador!!! *LA SOLUCIÓN ESTÁ EN TUS MANOS Y EN TUS HABITOS.*

¿Cómo usas tu tiempo en el estudio o trabajo?

Para hacer una buena gestión del tiempo ***es necesario tener presente, y saber la diferencia del significado de eficiencia, eficacia*** y ***el tiempo requerido*** para la realización de cada una de las tareas laborales, de estudios o lo que tengas que hacer.

La eficiencia consiste en hacer las cosas bien, pero no garantiza resultados. La eficiencia se centra en el método, en el cómo. Es decir, se limita a hacer bien lo que se hace.

La eficacia consiste en hacer lo que realmente hay que hacer. La persona eficaz sabe, ante todo, qué es lo que debe hacer. Tiene en mente los resultados que desea alcanzar y hace lo que le acerca a ellos. La persona que hace un buen empleo del tiempo es capaz de equilibrar eficiencia y eficacia

Eficacia vs Eficiencia

Por ejemplo, mientras una persona eficaz realiza 100 unidades de un determinado producto en 12 horas de trabajo, una persona eficiente tardaría 10 horas, al optimizar los recursos utilizados.

Recuerda: Siempre debes saber qué necesita o te pide la Gerencia y no solo debes realizar lo que te pide (ser eficaz o actuar con eficacia) sino darle un plus extra (ser eficiente o actuar con eficiencia) para sobresalir en lo que hagas.

Ejemplo 1: Juila y Gustavo

Una empresa de desarrollo de software contrata como pasantes a Julia y a Gustavo como desarrollador de software junior, ambos estudiaron en el mismo lugar. Gustavo realizó todas las actividades, tareas, prácticas y entregas que el curso le planteó en lenguaje Java con una aprobación del 100%. Julia por su parte también realizo todas las actividades, tareas, prácticas y entrega en lenguaje Java con una aprobación del 100% PERO con el adicional que por su cuenta realizó las mismas prácticas en varios lenguajes de programación (JavaScript, PHP, Python).

Cuando la Gerente del área de TI les asigna su primer tarea en lenguaje Python y dice que en 4 días máximo lo necesita, Gustavo debió aprender la sintaxis del lenguaje Python y terminó la tarea asignada al 4to día, es decir fue Eficiente al terminar la tarea asignada en tiempo estipulado. En cambio, Julia, en su caso al ya haber practicado con Python conocía la sintaxis del lenguaje y la tarea la finalizo al 2do día, es decir fue Eficaz al entregar en menos tiempo lo solicitado.

¿Dónde está la eficacia?

1. Julia al momento de estudiar, al hacer las prácticas con otros lenguajes de programación, fue eficaz en el estudio y de las prácticas.
2. Julia en el trabajo, pudo reducir el tiempo de entrega porque ya conocía el lenguaje, fue eficaz en la tarea asignada.

Ejemplo 2:

El líder de proyecto de TI les solicita a Julia y a Gustavo que realicen una programación de una función nueva en Java, que recupere los datos de la base de datos principal, los muestre en un reporte y lo necesita como máximo 3 días.

Tanto Julia como Gustavo tienen pleno conocimiento del lenguaje Java, Gustavo hace la programación perfecta y genera el reporte al tercer día, Julia también tardó exactamente el mismo tiempo con la diferencia que Julia agregó el logotipo de la empresa y además el reporte se ajustaba a los teléfonos móviles. Vale comentar que en las pruebas que se aplicó la programación de Gustavo tardaba menos tiempo en recuperar la información.

¿Dónde está la eficacia?

1. Julia, al momento de generar el reporte fue eficaz porque en el mismo tiempo pudo agregar el logo y hacer que se ajuste para los Teléfonos.
2. Gustavo, ha sido eficaz en la forma de programar porque su código tardó menos en recuperar la información.

La Ley de Parkinson

¿En qué consiste la Ley de Parkinson?

Un británico llamado Cyril Parkinson expuso en 1957 esta teoría tras observar su propia experiencia en una oficina pública británica. Su reflexión ayuda a entender por qué se posponen tareas y se pierde tanto el tiempo:

"*El trabajo se expande hasta llenar el tiempo disponible para que se termine*", lo que viene a decir que, cuanto menos tiempo se tiene para realizar una tarea, mayor es el esfuerzo para concluirla. Mientras que cuando se tiene más tiempo, menos esfuerzo se realiza para llevarla a cabo. A esto hay que sumarle que existe una tendencia a procrastinar las tareas y a ocupar el tiempo en otras tareas que distraen del objetivo principal.

Las **ideas claves que sostiene la Ley de Parkinson** pueden resumirse en tres:

1. **Trabajo:** lo que comentábamos anteriormente, que el trabajo se expande hasta ocupar el tiempo disponible para realizar la tarea.
2. **Tiempo:** el tiempo que se dedica a las diferentes tareas que se realizan a lo largo del día no es proporcional a su importancia.
3. **Gasto:** el gasto aumentará hasta cubrir la totalidad de los ingresos. Esta premisa es fundamental para entender por qué muchas personas no logran llegar a fin de mes o por qué una empresa gasta todos sus recursos disponibles. Se suele decir que las necesidades de una persona se invierten en base a sus ingresos.

¿Cómo aplicar la Ley de Parkinson en el trabajo?

Para aplicar la Ley de Parkinson lo primero que hay que hacer es: **marcarse un objetivo**. De no ser así, la tarea se dilatará en el tiempo. Además, existe una norma básica que sostiene que si una tarea requiere poco tiempo para realizarla hay que hacerla de inmediato sin planificación. De esta manera, se ahorra tiempo y

se reduce el número de tareas pendientes. Además, también ayudará a crear una rutina y adquirir hábitos de trabajo.

Técnica Pomodoro: basada en la filosofía del boxeo temporal, su objetivo es fijar un tiempo máximo para realizar tareas, tomar decisiones y lograr objetivos. Consiste en dividir el tiempo de trabajo en bloques de 25 minutos a los que se denominan “pomodoros”. Cuando se concluye un bloque, hay 5 minutos de descanso y tras 3 bloques de 25 se puede descansar entre 15 y 20 minutos. Durante el pomodoro no se permite ninguna distracción y en los minutos de descanso, no se permite ninguna tarea que requiera un esfuerzo mental.

A través de este método se pretende evitar el cansancio y la fatiga mental y fomentar la concentración y la agilidad mental.

Ley de Pareto o Regla 80/20: A esta regla se conoce como la Ley de Pareto. Vilfredo Pareto fue un economista y filósofo que expuso que el 80% de los resultados de una tarea proceden de hacer un esfuerzo correspondiente al 20% del tiempo dedicado. La clave de esta teoría está en concentrarse y prestar la máxima atención durante el 20% y en menor medida el 80% restante. De esta manera, se concentra un mayor esfuerzo en un período corto de tiempo y se logra diferenciar lo importante de lo que no lo es.

Evitar alargar el tiempo de trabajo: es importante ser estricto con el tiempo útil de trabajo para aprovecharlo al máximo y evitar postergar tareas. De este modo, se logra finalizar las tareas en menos tiempo y dedicándoles menos tiempo. Así se logra ser más eficiente y alcanzar las metas marcadas.

Aprovechar las horas de mayor productividad: cada persona rinde de una manera distinta y no tienen por qué coincidir en las mismas franjas horarias. Por ese motivo, es importante aprovechar las horas de mayor productividad para realizar las tareas y así lograr mejores resultados.

Evitar lo que te perjudica y anticiparse a aquellas conductas que conducen a la procrastinación, son las claves para hacer frente a la Ley de Parkinson en el trabajo o en cualquier otro ámbito. Para cumplir con los objetivos marcados se debe marcar prioridades y optimizar al máximo los espacios de tiempo que se dedican a realizar esas tareas evitando las distracciones. De esta manera, se reducirá el tiempo en llevarla a término.

¿Cuáles son algunos principios de la eficacia?

1. **Principio 1:** Se alargan las tareas al interrumpirlas. Las tareas se estiran y el tiempo no. Teóricamente, toda tarea requiere un tiempo determinado para obtener unos resultados -o calidad- deseados. Esta es, al menos, la impresión subjetiva que tenemos. Pero la realidad es muy distinta. Porque no nos es posible prácticamente nunca realizar de un tirón tareas medianamente complejas

2. **Principio 2:** Principio de la planificación. Cada minuto que inviertes en planificar tus actividades, te ahorra diez minutos de ejecución. El propósito de la planificación es llevarte a obtener el máximo "retorno en la inversión" que realizas en energía para tu vida personal y laboral.
3. **Principio 3:** Principio de las prioridades. Tu capacidad de fijar prioridades claras en el tiempo determina el avance hacia tus metas u objetivos.
4. **Principio 4:** Principio de las recompensas. Tus recompensas van a estar determinadas por tus resultados. La manera más efectiva de avanzar es lograr resultados de calidad en un mejor tiempo.
5. **Principio 5:** Principio de las secuencias. La administración eficaz del tiempo te permite manejar la secuencia de eventos.

Material complementario:

- http://193.147.134.18/bitstream/11000/7414/1/TFG-Ruiz%20Ruiz%2C%20Cristian.pdf

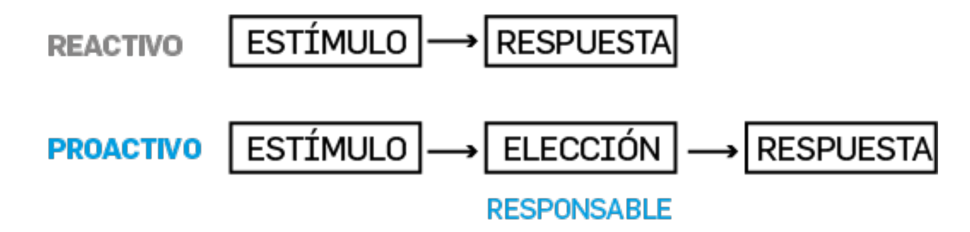
Modelo Estímulo - Respuesta

¿Vos de qué lado estas?

Según se responde ante los estímulos que se nos plantean en la vida, se actúa de manera reactiva o proactiva:

.- Reactivo: Las personas reactivas esperan a que las cosas se arreglen solas y no son capaces de tomar decisiones; responden ante estímulos siempre de la misma manera, impulsadas por emociones, sentimientos, circunstancias, condiciones, o por el mismo ambiente.

.-Proactivo: Los proactivos son capaces de romper ese modelo porque tienen libertad interior, es decir, tienen la capacidad de elegir reaccionar ante ciertas situaciones, pudiendo así, tomar decisiones más prudentes. Afrontan la vida personal y profesional tomando la iniciativa.



En el lenguaje :

El hombre es el único ser sobre la Tierra que tiene conciencia de sí mismo, lo que le dota de libertad. Y el uso de esta libertad es lo que le ha hecho progresar a lo largo de su historia. Tiene incluso libertad para elegir el contenido de su conciencia aún en medio de las circunstancias exteriores. Viktor Frankl (neurólogo y psiquiatra austriaco) pudo constatar como prisionero en un campo de concentración nazi, que quienes tenían más entrenada esta clase de libertad, fueron los que tuvieron más posibilidades de sobrevivir.

REACTIVO

- Culpan a los demás por sus actos
- Les suceden las cosas
- Esperan a que las cosas se arreglen solas
- Se ofenden fácilmente
- Se vuelven víctimas
- Se enfadan, pierden la cabeza, y dicen cosas que después lamentan
- Se quejan y lloran
- Se paralizan ante el frascaso y tienen miedo de volver a caer.

PROACTIVO

- Se hacen responsables de sus actos
- Hacen que als cosas sucedan
- Tienen iniciativa
- No se ofenden fácilmente
- Ejercen el control
- Piensan antes de actuar, son prudentes
- Vuelven a hacer el intento cuando algo sale mal
- Reconocen su error, aprenden de él y lo corrigen instantáneamente.

Lenguaje REACTIVO

- Yo soy así, no tengo remedio
- Intentaré
- No puedo, no tengo tiempo
- No puedo hacer nada
- Me vuelvo loco
- Tengo que hacerlo
- Debo
- Me arruinaste el día

Lenguaje PROACTIVO

- Puedo mejorar
- Lo haré
- Si, mañana a las 8:00am
- Examinemos nuestras opciones, debe hacer una solución.
- Controlo mis sentimientos y emociones
- Elegí hacerlo
- Prefiero
- No permitiré que tu mal estado de ánimo se me contagie.

MATRIZ DE ADMINISTRACIÓN DE TIEMPO. PRIORIZACIÓN

La utilización eficiente del tiempo se basa en la orientación de las tareas a la consecución de los objetivos. Por lo tanto, el establecimiento de prioridades es primordial a la hora de gestionar el tiempo. Priorizando, es decir, estableciendo un orden en las tareas diarias, se puede evitar la tiranía de lo urgente y ayuda a centrarse en lo importante. Para poder establecer una clara prioridad de las actividades planificadas, se deben clasificar estas como:

- Tareas urgentes e importantes: se trata de tareas prioritarias en cuanto a que están contempladas dentro de los objetivos, pero que, por falta de planificación o imprevistos de última hora, son urgentes y no se pueden demorar más.
- Tareas urgentes y no importantes: son tareas que han sido realizadas ya, pero que no son importantes puesto que su realización no aporta valor en el cumplimiento de los objetivos.
- Tareas no urgentes e importantes: son tareas contempladas como parte de los objetivos, las cuales se han sabido o pudieron planificar correctamente, con lo cual se pueden realizar con perspectiva.
- Tareas no urgentes y no importantes: son tareas que ni son importantes para el cumplimiento de los objetivos, ni son apremiantes.

En la siguiente Matriz de Administración del Tiempo, se muestran ejemplos de tareas / actividades tipo de cada uno de los cuadrantes mencionados:

	URGENTE	NO URGENTE
IMPORTANTE	I **ACTIVIDADES:** ▪ Crisis ▪ Problemas apremiantes ▪ Proyectos cuyas fechas vencen	II **ACTIVIDADES:** ▪ Prevención de crisis ▪ Preparación ▪ Construir relaciones ▪ Reconocer nuevas oportunidades ▪ Planificación
NO IMPORTANTE	III **ACTIVIDADES:** ▪ Interrupciones, algunas llamadas ▪ Correo, algunos informes ▪ Algunas reuniones ▪ Cuestiones inmediatas, acuciantes	IV **ACTIVIDADES:** ▪ Trivialidades, ajetreo inútil ▪ Algunas cartas ▪ Algunas llamadas telefónicas ▪ Pérdidas de tiempo ▪ Actividades de escape

Principales diferencias en la gestión del tiempo entre el alumnado de alto y bajo rendimiento.

	ALUMNADO DE **ALTO RENDIMIENTO**	ALUMNADO DE **BAJO RENDIMIENTO**
DISTRIBUCIÓN DEL TIEMPO	Desarrollan un patrón semanal de distribución del tiempo entre actividades de ocio, estudio y, en su caso, trabajo. *"voy adelantando cada día, o hasta los sábados [...] Después el resto de fin de semana me desconecto totalmente"*	Abordan actividades de distinta naturaleza sin ajustarse a rutinas temporales fijas. Tienden a procastinar. *"Lo primero que uno tiene que hacer es reconocer que uno no tiene como un horario para realizar las actividades"*
	Separan el tiempo dedicado al estudio del tiempo de ocio, concentrando éste al final de la semana. *"Estudio [durante la semana] para dejar el domingo con tiempo libre"*	Realizan tanto actividades de ocio como de estudio a lo largo de la semana. *"Y te dicen vamos a comer pancho y hablamos. Y cuando llegamos no hicimos la tarea, y uno no aprovecha el tiempo"*
PLANIFICACIÓN DE TAREAS	Realizan una previsión de tareas, las priorizan y las ubican en el tiempo disponible. *"Miro lo que me hace falta de trabajos y organizo para llegar al día"* *"Manejo las cosas que tengo a largo plazo para ir haciendo a poquitos"*	Planifican las tareas de manera menos sistemática, o bien no consiguen cumplir con la planificación. *"No llevo un planificación escrita [...] planifico como de un día para otro""Planifico qué debo hacer en la semana, en qué debo invertir mi tiempo, pero se me dificulta mucho porque dejo muchas cosas para último momemnto"*
HERRAMIENTAS PARA GESTIÓN DE TIEMPO	Amplia variedad de herramientas, incluyendo recursos tecnológicos. *"Hago como mapitas conceptuales, de lo que tengo pendiente"*	Menor variedad de herramientas, con predominio del uso de herramientas tecnológicas frente a las basadas en el lápiz y papel.
	Utilizo un cuaderno, lo tengo organizado por días [...] pongo, por ejemplo, tengo pendiente tal trabajo y tengo que investigar tal tema. *"En la noche organizo con papeles de colores las materias, si tengo que hacer un trabajo y para cuándo"* *"Utilizo el celular, para tal día tal tarea, y lo que tengo que hacer"*	*"Uso el celular, [...] y así controlo bien el tiempo"* *"En mi caso pongo alarmas en el celular"*
	Utilización de herramientas para la planificación y para comprobar el cumplimiento de tiempos y logro de metas. *"Coloco todo lo que tengo para la semana, y voy tachando y si alguna cosa me falta pongo un asterístico en rojo"*	Utilización de herramientas con sistemas de recuerdo, no integradas en una planificación de tareas. *"Uso el Celular, por que a mí se me olvidan mucho las cosas"*
VALORACIONES	Atribuyen valor a la gestipon del tiempo en la consecución del éxito académico. *"Pienso que la administración del tiempo es clave para hacer las cosas bien"*	Son concientes de su mala gestión del tiempo. *"Uno debe organizar el tiempo [...] La mayoría de estudiantes somos conscientes de que no administramos bien el tiempo"*







2021 Año de homenaje
al premio Nobel de medicina
Dr. César Milstein

Referencias:

- http://repositorio.uta.edu.ec/bitstream/123456789/32908/1/026%20ADP.pdf
- https://vinculando.org/educacion/herramientas-digitales-para-el-desarrollo-de-aprendizajes.html?utm_source=rss&utm_medium=rss&utm_campaign=herramientas-digitales-para-el-desarrollo-de-aprendizajes
- http://193.147.134.18/bitstream/11000/7414/1/TFG-Ruiz%20Ruiz%2C%20Cristian.pdf
- http://biblioteca2.ucab.edu.ve/anexos/biblioteca/marc/texto/AAR6907.pdf
- https://repository.ucc.edu.co/bitstream/20.500.12494/34408/1/2021_t%c3%a9cnica_empresa.pdf
- http://servicio.bc.uc.edu.ve/educacion/revista/48/art25.pdf

¿Qué es el desarrollo de software?

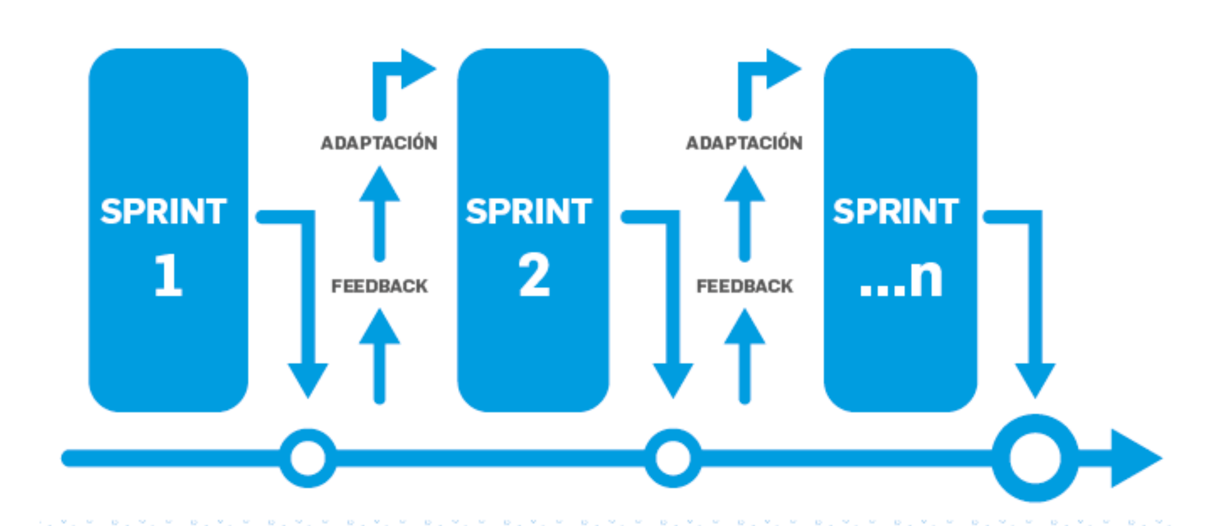
Desarrollo de Software

Según la IEEE , el software es el “*conjunto de programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación*”. Es la parte de un sistema que se puede codificar para ejecutarse en una computadora como un conjunto de instrucciones, e incluye la documentación asociada necesaria para comprender, transformar y usar esa solución. Estos documentos describen la organización del sistema, explican al usuario cómo utilizarlo y obtener eventuales actualizaciones del producto.

El software es un producto intangible, de alto contenido intelectual, que no sufre desgaste alguno y que puede ser potencialmente modificado de forma permanente. No se manufactura, sino que se desarrolla a través de proyectos que son realizados por equipos de personas altamente formadas. A pesar de la tendencia a desarrollar componentes que puedan ser reutilizados y adaptados a diferentes necesidades, la gran mayoría del software se construye a medida.

El proceso de desarrollo de software es un conjunto de actividades que da como resultado un producto que responde a las necesidades de un usuario. Este proceso define todas aquellas actividades necesarias para transformar los requisitos de un usuario en un producto. Estas actividades se las puede organizar con diferentes criterios y, como es un proceso intelectual-creativo, depende de las personas que formen parte del equipo y de sus decisiones y juicios. Existen varios procesos o modelos de desarrollo y podríamos catalogarlos en Secuenciales o Definidos, y en Empíricos. Los primeros responden siempre de la misma forma ante una determinada entrada, mientras que los segundos dependen de la evolución y adaptación de etapas anteriores.





El conjunto de fases (o procesos) por las que pasa el software desde que se concibe y se desarrolla hasta que finaliza su uso (retiro de servicio) se conoce como **ciclo de vida del software** . Estas fases están estandarizadas, existiendo un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, explotación y mantenimiento de un producto software, abarcando la vida del sistema, desde la definición de sus requerimientos hasta la finalización de su uso.

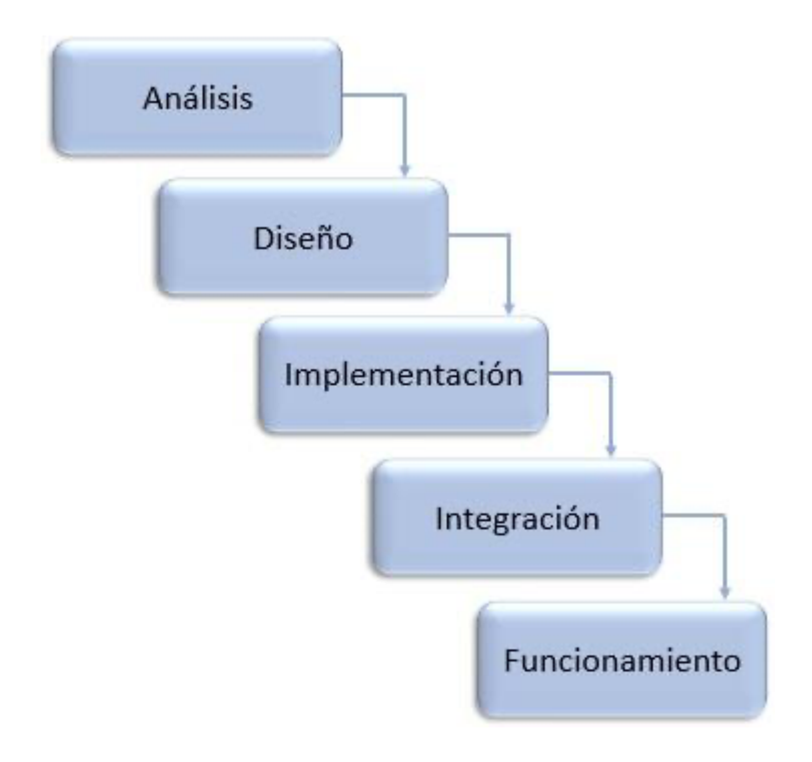
Metodología en Cascada

El modelo de desarrollo en cascada fue presentado por primera vez en 1970 por Royce . Se lo conoce también como modelo lineal secuencial. Su principal característica es que cada fase del desarrollo está bien definida y separada, siguiendo un orden secuencial en el que cada etapa dependiente de la finalización de la anterior y que esta última haya pasado un proceso de validación que apruebe el paso a la siguiente.

Comúnmente las etapas del modelo son:

1. Análisis y definición de los requerimientos.
2. Diseño del sistema y del software.

3. Implementación y pruebas unitarias.
4. Integración y pruebas de sistema.
5. Funcionamiento y mantenimiento.



Fuente: Ingeniería del software . Sommerville, I. Capítulo 4.

Imaginemos que somos un equipo de desarrollo de software y recibimos un pedido de la empresa “ArgProg”. Se trata de una PyME nacional del rubro farmacéutico que, en los últimos años, experimentó un importante crecimiento gracias a su capacidad de idear soluciones innovadoras a problemas de la industria. Como resultado, la empresa incrementó tanto su plantilla de empleados como su cartera de proyectos. En este nuevo contexto, “ArgProg” nos demanda el desarrollo de un software que mantiene conectadas a las personas y a los proyectos, ya que esa interacción es la base de la innovación permanente.

Si bien es un gran desafío, conocemos el recorrido tradicional para responder a esta demanda:

1. nos reunimos internamente para decidir si tomamos el proyecto,

2. invertimos tiempo y recursos en diseñar hasta el último detalle de la solución que vamos a ofrecer a la empresa (alcance del proyecto),

3. listamos y ordenamos todas las tareas que serán necesarias para construir esa solución (descomposición de la estructura de trabajo)

4. estimamos cuánto tiempo y recursos implicará la realización de esas tareas (cronograma y presupuesto del proyecto). Tomamos como base para esta estimación los proyectos similares que hicimos en el pasado.

Una vez que plasmamos toda esta información en documentos (planificación), se lo presentamos a la empresa y, si logramos un acuerdo, entonces sí:

5. Ejecutamos las tareas planificadas (ejecución)

6. Monitoreamos el progreso para evitar desvíos respecto de lo que habíamos planificado en los documentos (monitoreo y evaluación)

7. Completamos el desarrollo del software, se lo entregamos a nuestro cliente y ¡finalizamos el proyecto! (cierre).

Esta metodología para gestionar proyectos, denominada “metodología tradicional”, parece clara y relativamente sencilla de implementar ¿no?
Sin embargo, según datos del Standish Group, más del 70% de los proyectos de desarrollo de software gestionados bajo esta metodología no tienen éxito.

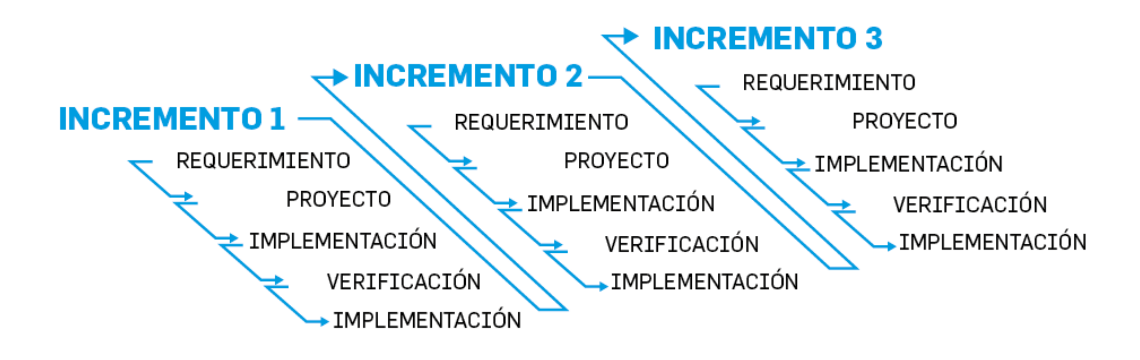
Las principales razones:

- Suponen que los usuarios del software tienen completamente clara su necesidad o deseo y esto muchas veces no ocurre en la realidad. En el ejemplo, “ArgProg” busca “mantener conectados a las personas y a los proyectos”, sin embargo ¿qué implica estar “conectados”? es algo difuso... ¿Alcanza con que sepan qué están desarrollando los otros equipos? ¿Lo relevante es que estén al tanto de los desafíos que enfrentan sus compañeros? ¿Aportar sus experiencias o saberes para ayudarlos a resolverlos? A esta falta de claridad sobre la necesidad hay que sumarle otro factor: la multiplicidad de actores involucrados. La empresa está conformada por un directorio, gerentes, mandos medios, trabajadores de línea, cada uno de ellos con sus propios intereses,
- Suponen que el equipo de desarrollo puede crear la solución ideal y describirla de forma precisa y detallada antes de empezar a construirla. En el caso de “ArgProg” implicaría tener absoluta certeza sobre la arquitectura que tendrá el software, cada aspecto de su funcionamiento, cada una de las funcionalidades que incluirá, cada detalle del “look and feel”, etcétera, antes de escribir el primer código. Idear una solución y plasmarla en documentos con este nivel de detalle es, en la mayoría de los casos, inviable y lleva a que se desarrollen más funcionalidades de las deseadas o muchas menos de las necesarias, a que se sobreestimen aspectos que no agregan valor al cliente y se descarten aquellos que fueron realmente valiosos.
- Dada la claridad de las necesidades y la capacidad de idear una solución que las satisfaga, las metodologías tradicionales suponen que es posible planificar todos los aspectos del proyecto (actividades, cronograma y presupuesto) y ajustarse a ese plan. El problema es que esto sólo es viable cuando contamos con experiencias anteriores en desarrollos similares, las cuales podemos usar

como referencia para estimar el esfuerzo que demandará el proyecto actual. Esto no es viable cuando apuntamos a generar soluciones innovadoras para las cuales no contamos con antecedentes para planificar con certeza.

Metodología iterativa e incremental

La metodología iterativa e incremental deriva del proceso de desarrollo en cascada, pero, con la diferencia de que aquí se admite que las etapas se solapan en tiempo con la finalidad de flexibilizar el tiempo de desarrollo total y así poder alcanzar resultados funcionales de manera temprana.



Referencia: http://modelado-de-la-informacion.blogspot.com/2015/09/metodologia-de-desarrollo-iterativo-y.html

Esta metodología involucra dos procesos fundamentales:

- El proceso **incremental**: con esto se busca desarrollar una parte del producto que se pueda integrar al conjunto a medida que se alcanza un grado de completitud.

- El proceso **iterativo**: se realiza en ciclos donde se revisa y mejora el producto. De manera que la calidad del producto aumente, y no siempre implica la integración de nuevas funcionalidades.

Procesos Ágiles

A fin de superar las dificultades o límites de las metodologías tradicionales, surgieron las metodologías ágiles para la gestión de proyectos. Los métodos o procesos ágiles consideran un enfoque iterativo para las etapas de especificación, desarrollo y entrega del software. Estos métodos fueron pensados para el

desarrollo de aplicaciones donde los requerimientos del sistema cambian rápidamente y también están enfocados en poder entregar software funcional de forma rápida a los usuarios, quienes pueden evaluar y proponer nuevos requerimientos o cambios para iteraciones próximas.

Estos procesos se basan en los valores y los principios definidos en el Manifiesto Ágil elaborado en 2001 por un grupo de profesionales críticos de los modelos de desarrollo de software propuestos hasta el momento a los que consideraban excesivamente pesados y rígidos por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo.

Los valores que propone este manifiesto son:

- Valorar más a los individuos y sus interacciones que a los procesos y las herramientas
- Valorar más el software funcionando que la documentación exhaustiva
- Valorar más la colaboración con el cliente que la negociación contractual
- Valorar más la respuesta ante el cambio que seguir un plan

También se ven los siguientes 12 principios:

1. Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
3. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los períodos breves.
4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
5. Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
6. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
7. El software que funciona es la principal medida del progreso.
8. Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica enaltece la agilidad.
10. La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.

12. En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

Si bien hay diferentes métodos ágiles, todos comparten algunos **principios elementales**:

- **Participación del cliente**: Los clientes deben estar implicados en todo el proceso de desarrollo, su rol está enfocado en proporcionar y dar prioridad a nuevos requerimientos del sistema, como así también participar en la evaluación de los entregables de cada iteración.
- **Entrega incremental**: El software se desarrolla en incrementos y el cliente es quien determina qué necesita incluir en cada incremento.
- **Enfocado en personas, no en procesos**: Hay que identificar y explotar las habilidades de cada una de las personas del equipo. Cada equipo debe desarrollar sus propias formas de trabajar, es decir, sin imponer procesos formales.
- **Aceptar el cambio** : Siempre se debe tener en cuenta que los requerimientos del sistema pueden cambiar, y por lo tanto el diseño debe contemplar esto.
- **Mantener la simplicidad** : Se debe enfocar la simplicidad tanto en el software que se desarrolla como en el proceso de desarrollo.

Existen diferentes metodologías ágiles que, si bien comparten los mismos principios, cada una posee sus propias características que las diferencian entre sí. Algunas de ellas son la Programación Extrema (XP, Extreme Programming), el Desarrollo Rápido de Aplicaciones , el Desarrollo de prototipo, y Scrum.

¿Que es Scrum?

Melé

SCRUM fue creado por Ken Schwaber y Jeff Sutherland a partir de la reflexión sobre su propia práctica y la de sus colegas, y actualmente es el enfoque más utilizado a nivel mundial para el desarrollo de software. Es especialmente útil para trabajar en entornos complejos en los cuales tanto la necesidad a cubrir como la solución que la satisfaga no son claras de antemano y sólo pueden conocerse a medida que se construye y se pone a prueba el software.

Scrum es un marco de trabajo a través del cual las personas pueden abordar problemas complejos adaptativos, a la vez que se entregan productos de forma eficiente y creativa con el máximo valor. En scrum se aplica un conjunto de buenas prácticas para trabajar de manera colaborativa y obtener así el mejor resultado posible en proyectos complejos que demandan constantes cambios y adaptaciones.

La palabra scrum tiene su origen en un ámbito alejado al de la gestión de proyectos: el deporte. En rugby, “Scrum” es el término que define a la formación en la que ambos equipos empujan de forma coordinada para obtener la pelota. Esta

formación requiere a cada equipo empujar no sólo en la misma dirección sino al mismo tiempo para lograr la suma de las fuerzas individuales. Representa un claro ejemplo de trabajo en equipo y coordinación entre todos los miembros de un mismo equipo.

Uno de los pilares de Scrum es la **transparencia**. Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. Esto requiere que dichos aspectos sean definidos en base a un estándar común, de tal modo que los observadores compartan un entendimiento común de lo que están viendo. Por ejemplo:

- Deben compartir un lenguaje común todos los participantes para referirse al proceso; y,
- Aquellos que desempeñan el trabajo y quienes inspeccionan el incremento deben compartir una definición común de “Terminado”.

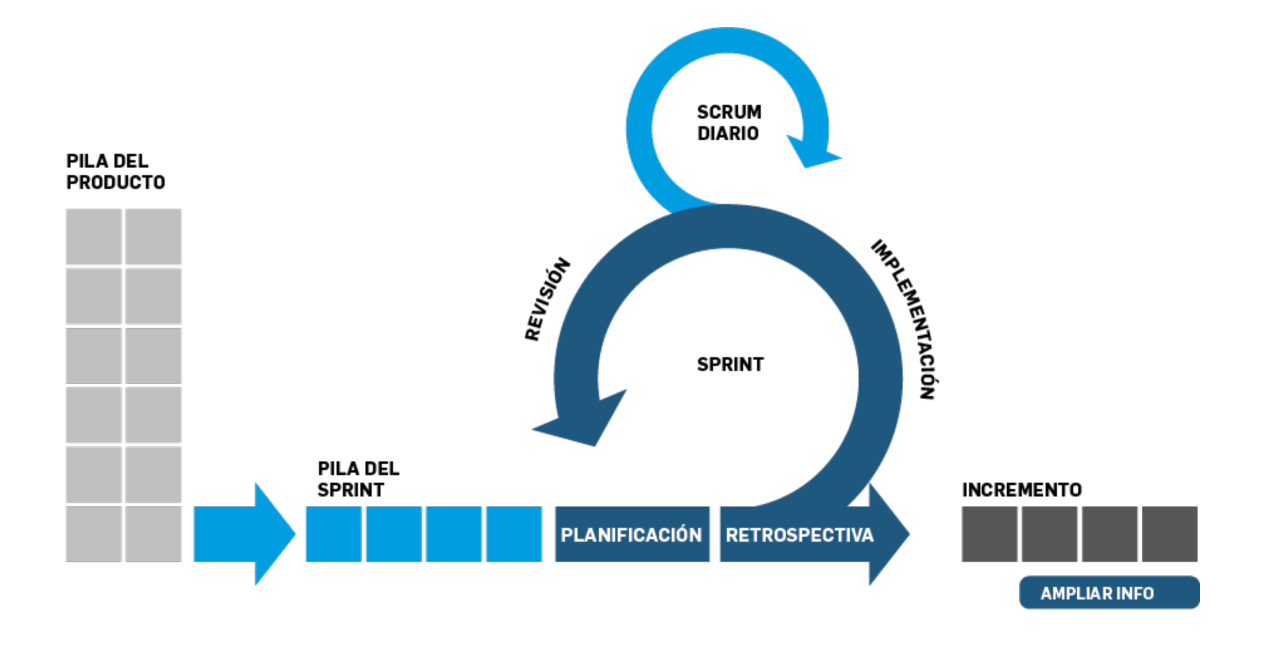
Otro de los pilares es la **inspección**. Los usuarios de Scrum deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo para detectar variaciones indeseadas. Su inspección no debe ser tan frecuente como para que pueda interferir en el trabajo. Las inspecciones son más beneficiosas cuando se realizan de forma diligente por inspectores expertos en el mismo lugar de trabajo.

Finalmente, otro pilar de la metodología es su **adaptación**. Si un inspector determina que uno o más aspectos de un proceso se desvían de los límites aceptables y que el producto resulta inaceptable, el proceso o el material que está siendo procesado deben ajustarse. Dicho ajuste deberá realizase cuanto antes para minimizar desviaciones mayores.

Desde la perspectiva de SCRUM, un proyecto se considera exitoso si entrega el mayor valor posible para el usuario en un tiempo dado. ¿Cómo sabemos qué aporta valor al o los usuarios? A medida que se va construyendo y testeando el software se va comprendiendo más profundamente la problemática y la necesidad de los usuarios y las posibles formas de satisfacerlo (incluso los mismos usuarios al interactuar con el producto van terminando de comprender su propia necesidad).

Los proyectos se llevan a cabo en equipos que trabajan en ciclos temporales cortos y duración fija. Cada ciclo se conoce como Sprint y tiene que proporcionar un resultado completo, un incremento del producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando este lo solicite. Los componentes del ciclo son:

- **Eventos**: Planificación del sprint, Scrum diario, Revisión del sprint, y Retrospectiva del sprint
- **Roles**: Desarrollador, Propietario del producto, y Scrum master
- **Artefactos**: Pila del producto, Pila del sprint, e Incremento.



Ciclo de desarrollo (Sprint) siguiendo metodología Scrum

Eventos - Planificación en Scrum

Eventos

El núcleo de esta metodología son los Eventos. Cada uno de estos, se enmarca en el ciclo iterativo denominado Sprint y hace que sea posible llegar a un entregable útil al final de cada iteración. **Los *Sprints*** son el corazón de la metodología, **son períodos de tiempo fijos**, generalmente de un mes o menos, y comienzan inmediatamente después de finalizado el *Sprint* anterior. Durante un *Sprint* no se pueden hacer cambios que perjudiquen los objetivos de este, se refina la pila de producto y eventualmente se podría clarificar o renegociar el alcance con el propietario del producto a medida que se aprende sobre el avance. Solo si el objetivo se vuelve obsoleto, el propietario del producto tiene la autoridad para cancelar el *Sprint* .

Planificación de Sprint

La planificación es el evento que da inicio al *Sprint* y es donde se lleva a cabo la organización del trabajo a realizar. Este evento no es necesariamente exclusivo del equipo de *Scrum* , sino que se puede invitar a otras personas para que aconsejen en los temas pendientes.

Durante la planificación se tratan los siguientes temas:

- ¿Por qué es valioso este *Sprint* ?

El propietario del producto propone cómo él va a aumentar su valor y utilidad con el *Sprint* . Todo el equipo colabora para definir el objetivo del *Sprint* que transmite

el por qué es valioso para los interesados en el producto. Al final de este evento el objetivo debe estar bien definido.

- ¿Qué se puede hacer en este *Sprint* ?

A lo largo de la discusión, los desarrolladores eligen ítems de la pila de producto, para incluirlos en el *Sprint* en curso. Durante este proceso, el equipo puede refinar estos ítems mejorando el entendimiento sobre cada uno. Este tema, de conocer cuánto se puede hacer, los desarrolladores lo entrenan teniendo en cuenta su rendimiento pasado, la capacidad y su definición de Terminado.

- ¿Cómo se llevará a cabo el trabajo seleccionado?

Para cada uno de los elementos seleccionados, los desarrolladores planean el trabajo necesario para crear un incremento que cumpla con la definición de Terminado. La manera en que estos lo ejecutan queda a su discreción y suelen descomponer los ítems de la pila de producto en tareas más pequeñas, de un día o menos.

Scrum Diaria

Las *Scrum* diarias (Daily Scrum) son reuniones de no más de 15 minutos, en las cuales se sincroniza el trabajo desarrollado y se establece el plan para el día en curso. Generalmente se recomienda hacerlas viendo el tablero con la pila de sprint. En las reuniones participan los desarrolladores, pero también pueden participar otras personas relacionadas con el proyecto o la organización, aunque normalmente estas no intervienen. Se comparten los avances, los problemas y las ideas, también si es necesario se actualiza la pila de *Sprint*. Es importante tener en cuenta que no es una reunión de inspección o control, sino que de comunicación entre el equipo.

Revisión de Sprint

Sobre el final del *Sprint* se hacen las reuniones de revisión de *Sprint* para comprobar el incremento. Estas suelen durar entre una y dos horas, en caso de que el incremento tenga una complejidad muy alta, podrían extenderse aún más. Estas pueden permitir al equipo y al propietario del producto tomar sensibilidad del ritmo de construcción y la trayectoria que cobra la visión del producto. Son instancias en las que se reajusta la pila de producto para incluir nuevas oportunidades.

Retrospectiva de Sprint

El propósito de las reuniones de Retrospectiva de *Sprint* es planear nuevas formas de mejorar la calidad y efectividad. Se analiza lo que sucedió en la última Sprint con respecto a los individuos, sus interacciones, los procesos, lo que fue bien y los problemas que se encontraron, los que se resolvieron y los que quedaron pendientes.

Roles - equipo SCRUM

Roles / Equipo

La unidad fundamental de Scrum es un equipo de personas (Scrum *Team*). Cada equipo consiste en un *Scrum Master*, un propietario del producto (*Product Owner)* y desarrolladores (*Developers*).

El equipo debe ser suficientemente pequeño como para ser ágil y grande como para completar una cantidad considerable de trabajo dentro de un *Sprint*. Normalmente los equipos están conformados por 10 personas o menos, ya que en general, los equipos pequeños se comunican mejor y son más productivos.

El equipo es responsable de todas las actividades relacionadas al producto desde la colaboración con las partes interesadas, verificación, mantenimiento, experimentación, desarrollo y cualquier otra cosa que pueda ser requerida.

Desarrolladores

Los desarrolladores son aquellos que producen cada uno de los incrementos de cada *sprint* y siempre son responsables de:

- Crear un plan para el *Sprint*, la pila de *Sprint* (*Sprint Backlog*).
- Asegurar la calidad mediante la adopción de una Definición de Terminado.
- Adaptar el Plan de Cada Día para conseguir el Objetivo del *Sprint* (*Sprint Meta*).

Propietario del Producto

El Propietario del Producto es el responsable de Gestionar Efectivamente la Pila de Producto (*Product Backlog*), Lo que incluye:

- Desarrollar y Comunicar el objetivo m del Producto (*Meta del producto*).
- Crear y comunicar cada uno de los artículos de la pila de producto.
- Ordenar los artículos de la pila de producto siguiendo alguna prioridad.
- Asegurar que la pila de producto sea transparente, visible y comprensible.

Es importante resaltar que el propietario del producto es una única persona, y por tanto, se deben respetar sus decisiones. Estas, deben ser reflejadas en el contenido y orden en la pila de producto, como así también en el incremento y la revisión del *Sprint*.

Scrum Master

El *Scrum Master* es el encargado de asegurar que la metodología se lleve a cabo tal y como está definida. Esto se logra haciendo que cada persona del equipo entienda la teoría y práctica de la metodología.

La persona con el rol de *Scrum Master* sirve al equipo de Scrum de varias formas:

- Orienta a los miembros del equipo en la autogestión y la multifuncionalidad.
- Ayuda a los equipos a enfocarse en crear incrementos de alto valor que cumplan con la definición de Terminado.
- Remueve los impedimentos de progreso que pudiera tener el equipo.
- Asegura que todos los eventos de *Scrum* se lleven a cabo y que sean positivos, productivos y dentro de los tiempos establecidos.

El *Scrum Master* también colabora con el propietario del producto de las siguientes maneras:

- Ayuda a encontrar técnicas para definir efectivamente el objetivo del producto y la gestión de la pila de producto.
- Ayuda a que los miembros del equipo entiendan los artículos de la pila de producto.
- Ayuda a establecer un planeamiento del producto de manera empírica para entornos que sean complejos.

Finalmente, también colabora con el resto de la organización con actividades como:

- Liderar, planear y entrenar a los miembros de la organización en lo que respeta a la adopción de *Scrum*.
- Planear y aconsejar cómo implementar *Scrum* dentro de la organización.
- Ayudar a los empleados a comprender y llevar a cabo la práctica de la metodología, mediante un enfoque empírico para trabajos complejos.

Durante la reunión de planificación de *Sprint*, el *Scrum Master* deberá desempeñar las siguientes funciones:

- Realizar esta reunión antes de cada *Sprint*.
- Asegurar que se cuenta con una pila de producto preparada por el propietario del producto.
- Ayudar a mantener el diálogo entre el propietario del producto y los desarrolladores.
- Asegurar que se llegue a un acuerdo entre el propietario del producto y los desarrolladores con respecto a lo que incluirá el incremento.
- Ayudar a comprender la visión y las necesidades de negocio del cliente.
- Asegurar que se ha realizado una descomposición y estimación del trabajo realistas.
- Asegurar que al final de la reunión estén determinados los siguientes puntos:

    1. Los ítems de la pila de producto que se van a ejecutar
    2. El objetivo del *Sprint*.
    3. La pila de *Sprint* con todas las tareas estimadas.
    4. La duración del *Sprint* y la fecha de reunión de revisión.

5. La definición de Terminado que determinará que el incremento está listo.

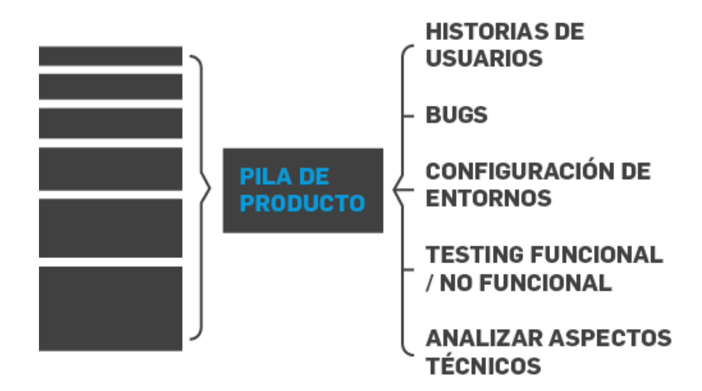
Artefactos SCRUM

Artefactos

Pila de Producto

La Pila de Producto (*Product Backlog*) es una lista ordenada. Cada una de las entradas de esta lista son posibles trabajos a seleccionarse para su realización durante una reunión de planificación de *Sprint*. El refinamiento de los ítems de la pila de producto es el acto de convertir esos ítems en elementos más detallados y precisos, en este proceso también se les asigna una prioridad y un “tamaño”.

La pila de producto puede incluir ítems para explorar las necesidades del cliente, analizar opciones técnicas, y otros ítems de trabajo tales como la corrección de errores (bugs) o la configuración del entorno. Todo lo que esté en la pila de producto representa o aporta a conseguir el objetivo del producto (*Product Goal*) y este último es el objetivo que el equipo tiene que conseguir cumplir a largo plazo.



Fuente: Tipos de elementos en la pila de producto

Pila de Sprint

Este artefacto, pila de *Sprint* (*Sprint Backlog*), delimita el trabajo necesario para alcanzar determinado incremento, y sirve para marcar el avance. También sirve como herramienta de comunicación del equipo. Esto es visible para todos y representa una imagen en tiempo real del trabajo que llevan los desarrolladores para alcanzar el Objetivo del *Sprint* (*Sprint Meta*). Este último se crea durante la reunión de planificación de *Sprint* y a lo largo de todo el ciclo, los desarrolladores lo tienen presente.

Incremento

Un incremento de producto es un paso concreto que acerca el desarrollo al objetivo del producto. Cada incremento agrega valor al incremento anterior y bajo

un proceso de verificación, se asegura que todos los incrementos trabajen bien juntos.

Existe un compromiso entre el incremento y la Definición de Terminado (*Definición de Hecho*). Esta definición es una descripción formal del estado que debe alcanzar el incremento para cumplir con el nivel de calidad requerido para el producto. Si por alguna razón algún ítem de la pila de producto no cumple con la definición de terminado, no debe formar parte del entregable y ni siquiera ser presentado en la reunión de revisión de Sprint. En lugar de esto, debe volver a la pila de producto para ser considerado en un futuro.

SCRUM

MARCO DE TRABAJO

PILA DEL PRODUCTO

PO

SM TD

SCRUM DIARIO

Revisión del trabajo resolución de impedimientos

Presentación del incremento. Se escuchan sugerencias. Se responden preguntas

REVISIÓN

IMPLEMENTACIÓN

SPRINT

SM TD S

PLANI FICACIÓN

PILA DEL SPRINT

PLANIFICACIÓN

RETROSPECTIVA

INCREMENTO

Exposición de prioridades. Resolución de dudas

PO SM TD

Estimación del esfuerzo para cada requerimiento.

PO SM TD

Revisión del sprint con objeto de mejora continua

ROLES

PO (Product Owner)
Quien determina prioridades.
Una sola persona.

SM (Scrum Master)
Gestiona y facilita el proceso Scrum.

TD (Team Developer)
Quienes construyen el producto.

S (Stackholders)
Interesados, usuarios.

ARTEFACTOS

Product Backlog
Listado de requisitos del producto. Sin mucho detalle. Priorizados.
El PO es el responsable y quién decide.

Sprint Backlog
Requisitos comprometidos por el equipo para el sprint. Deben tener suficiente detalle para su ejecución.

Product Increment
Parte del prodcuto desarrollada en el Sprint.

CEREMONIAS

Planning
Jornada de trabajo. El propietario define prioridades y responde dudas. El equipo estima el esfuerzo y se elabora el sprint backlog. Se definen objetivos y tareas.

Daily
15min. de duración. DIrigida por el SM. Se actualiza el sprint backlog.

Review
Informativa. 4 hs aporx. moderada por el SM.

Retrospective
Apunta a la mejora continua.

SPRINT

Ciclo de desarrollo que tiene por objeto un incremento al producto operativo. Tine una duración de 30 días como máximo.

VALORES

- Valorar a los individuos por encima de los procesos.
- El software que funciona por encima de la documentación exhaustiva.
- La colaboración del cliente por encima de la negociación contractual.
- La respuesta al cambio por encima del seguimiento de un plan.

Especificaciones y requerimientos

Es importante realizar una especificación de los requisitos, es decir documental de forma completa, precisa y verificable los requisitos, el diseño y el comportamiento u otras características de un sistema o componentes de este.

Requerimientos Funcionales

Los requisitos funcionales son declaraciones de los servicios que deben proporcionar el sistema. Describen cómo debe reaccionar el sistema a entradas particulares o cómo debe comportarse bajo determinadas condiciones.

Por tanto, la especificación de requerimientos debería cumplir las características de ser completa y consistente. Para que sea completa, todos los servicios descritos por el usuario deben estar definidos. Por otra parte, para que sea consistente los requerimientos no deben ser contradictorios.

Requerimientos no Funcionales

Los requisitos no funcionales son aquellos que se utilizan a las propiedades que deben tener, como fiabilidad, capacidad de almacenamiento, tiempo de respuesta, etc.

Generalmente surgen de necesidades del usuario que políticas tienen que ver con restricciones de presupuesto, la interoperabilidad con otros sistemas, factores externos, regulaciones, privacidad, seguridad, etc.

Requerimientos en Scrum y Estimación de tiempos

Dentro de las Metodologías Ágiles se Suelen Como utilizar las Historias de Usuario (*User Stories*) Como Herramienta para Definir los Requerimientos del Sistema. Estas son descripciones o especificaciones de una función, validadas por un usuario o cliente del sistema.
Generalmente las historias se escriben en un lenguaje que el usuario pueda entender y que refleje una descripción sintetizada de lo que este desea. En lo posible se debe tratar de eliminar ambigüedades y malas interpretaciones.

Para redactar una historia de usuario se tiene que seguir una estructura como la siguiente:



Escribir las historias de usuario con este formato tiene ciertas ventajas:

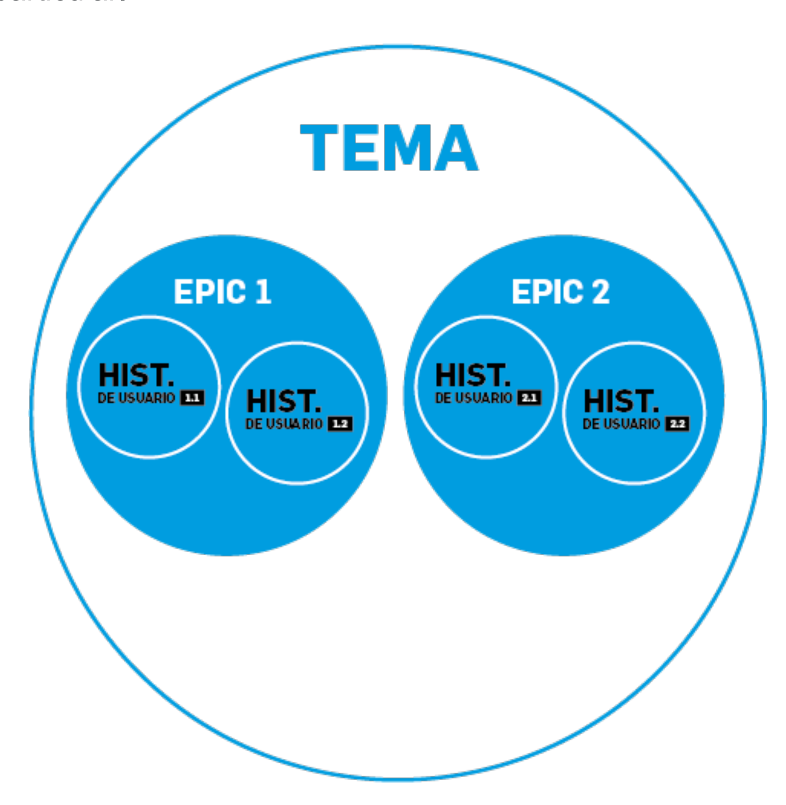
- Primera persona: Invita a quien la lee a ponerse en el lugar del usuario.

- Priorización: Ayuda al propietario del producto a priorizar y visualizar mejor cuál es la función, quién se beneficia y cuál es el valor.
- Propósito: la presencia del propósito de una función brinda al equipo la posibilidad de plantear alternativas que cumplan con el mismo objetivo en caso de que el costo de la función sea demasiado alto o su construcción no sea viable.

Podemos redactar la historia de usuario del siguiente ejemplo: “Daniel llegó hoy a Neuquén, pero está perdido en su automóvil y dio vueltas por horas. Quiere llegar al museo provincial de bellas artes “Museo Nacional de Bellas Artes de Neuquén”. En su celular deberá buscar cómo llegar si la calle es Mitre y Santa Cruz. ”
Utilizando la estructura propuesta de la siguiente forma:
Como *conductor* quiero *buscar un destino a partir de una calle y altura* para poder *llegar al lugar deseado*.

Temas, Épicas e Historias

Durante el proceso de análisis de historias de requerimientos se suele agrupar a las últimas de usuario en épicas ya su vez, estas últimas están relacionadas con un tema en particular.



Relación entre temas, épicas e historias de usuarios

Usualmente las historias de usuario se ubican en la pila del producto, son derivadas de los temas y agrupadas en épicas, luego se traducen en tareas en la pila de *Sprint*.

VISIÓN | TEMAS | PRODUCT BACKLOG | EPICAS HISTORIAS DE USUARIOS | SPRINT BACKLOG | TAREAS

Historias de usuario en SCRUM

Método INVEST

Este método, desarrollado por Bill Wake en 2003, permite asegurar la calidad de la escritura de las historias de usuario. Para esto debe cumplir con las siguientes características:

- Independiente: Cada historia de usuario puede ser planificada e implementada en cualquier orden. No depende unas de otras, si esto ocurre se deben dividir o combinar.

- Negociable: Las historias deben ser negociables y sus detalles serán acordados por el cliente / usuario y el equipo durante la fase de «conversación».

- Valiosa: Una historia de usuario tiene que ser valiosa para el cliente o el usuario.

- Estimable: Una buena historia de usuario debe ser estimada con la precisión suficiente para ayudar al cliente o usuario a priorizar y planificar su implementación. Si no podemos estimarla debemos incidir en la fase de conversación o dividirla.

- Pequeña (Small): Pequeñas. Solemos hacerlas de tal modo que ocupen como máximo un sprint.

- Comprobable (Testable): La historia de usuario debe poderse probar (Hemos trabajado con anterioridad en la fase "confirmación" de la historia de usuario. Tanto el usuario como el equipo de desarrollo tienen que poder probarla para saber cuándo está finalizada.

Método SMART

Cuando se descomponen las historias de usuario en tareas, se puede utilizar el método SMART, un método que sirve para evaluar si las tareas están correctamente definidas. SMART es un acrónimo (en inglés) de las siguientes características:

- Específico: Una tarea debe ser lo suficientemente específica para que todos puedan entenderla.
- Medible: ¿Hace lo que se pretende? Es el equipo quien debe ponerse de acuerdo sobre lo que esto significa.
- Alcanzable: ¿Es razonable la meta? Consejo: Es bueno establecer metas que representen un desafío, pero puede ser contraproducente no alcanzarlas.

- Relevante: Cada tarea debe ser relevante, contribuyendo a la historia en cuestión.
- A tiempo (T*ime-Boxed*): Una tarea debe estar limitada a una duración específica. No necesariamente debe ser una estimación formal en horas o días, pero debe haber una expectativa para que la gente sepa cuándo debe ayudar.



Características del método SMART.

Subdivisión de Historias de Usuario

Existe una guía de patrones para subdividir historias de usuario grandes o complejas en otras más pequeñas y alcanzables. Esta técnica de subdivisión de historias de usuario se puede llevar a cabo mediante juegos de cartas como el disponible en este enlace .

#SlicingPatterns

Una guía de patrones para subdividir User Stories grandes o complejas

PARETICEMOS: realizar primero el 20% del trabajo que agregue el 80% del valor. Evaluar si es necesario implementar la parte restante.

CAMINO ASFALTADO: implementar primero el camino principal y luego los caminos alternativos y de error en User Stories adicionales.

DIFERIR REQUERIMIENTOS NO FUNCIONALES: implementar la y agregar en User Stories adicionales requerimientos no funcionales: performance, seguridad, escalabilidad, accesibilidad, usabilidad, etc.

LO PRIMERO Y LO ÚLTIMO: en un proceso de varios pasos, implementar el primero y el último. Agregar los pasos intermedios en otras User Stories.

MÁS CRITERIOS: por tipo de usuario, por interesado, por distintas reglas de negocio, por interfaz gráfica, por browser, por usabilidad, por distintas formas de ingreso de datos, por distintas formas de mostrar los datos, por escenarios de prueba o distintos criterios de aceptación.

Patrones de división de historias de usuario.

Las 3 C de las Historias de Usuario

Una historia de usuario está compuesta de tres elementos que son fundamentales. La trivial es la primera y **la Tarjeta** (Card), es Donde SE ESCRIBE La historia de Manera Clara y con la ambigüedad minimum. El segundo componente es la **Conversación**, es importante debatir y validar con el cliente y el equipo de desarrollo, por lo general, esto se realiza en la reunión de planificación. Y finalmente la **Confirmación**, es importante confirmar que todos los involucrados han comprendido los requisitos.

Puntos de historia

"El trabajo necesario para realizar un requisito o una historia de usuario no se puede prever de forma absoluta porque rara vez son realidades de una solución única. En el caso de que se pudiera, por otra parte, la complejidad de la medición haría una métrica demasiado pesada para la gestión ágil.

No resulta posible estimar con precisión la cantidad de trabajo que hay en un requisito. En consecuencia, tampoco se puede saber con antelación cuánto tiempo exigirá, porque a la incertidumbre del trabajo se suman las inherentes al tiempo: no se puede estimar la cantidad o la calidad del trabajo que realiza una «persona media» por unidad de tiempo, porque son muy grandes las diferencias de unas personas a otras. Es más: la misma tarea realizada por la misma persona requerirá diferentes tiempos según las 56 circunstancias.

Por todas estas razones, al estimar de forma ágil, se prefiere emplear unidades relativas. En gestión ágil se suelen emplear «puntos» como unidad de trabajo, utilizando denominaciones como «puntos de historia». Cada organización, según sus circunstancias y su criterio, institucionaliza su métrica de trabajo, su «punto». Es el tamaño relativo de tareas que se suele emplear. Es importante que el significado y la forma de aplicar la métrica sea siempre la misma en las mediciones de la organización, y que sea conocida por todos. El tipo de «punto» depende de la organización. En un equipo de programación el punto puede ser equivalente a preparar una pantalla de inicio de sesión; para un equipo de diseño gráfico, la maquetación de un tríptico.

El «punto» ayuda, por un lado, a dimensionar la estimación de una tarea comparándola con una ya conocida, y por otro lado, a contrastar la dificultad que la tarea presenta para cada miembro del equipo según sus especialidades. Un ejemplo para ilustrar esto último podría ser el esfuerzo que cuesta freír un huevo. Si se estima cuántos «huevos fritos» costaría planchar una camisa, la respuesta dependiente de la persona. Alguien puede ser muy habilidoso friendo huevos, pero muy torpe para planchar camisas, y estimará que eso le costaría «8 huevos fritos»; es decir, «8 puntos». Alguien muy acostumbrado a las tareas domésticas, en cambio, podría estimar la tarea en «un punto» o «un huevo frito».

Ambos tienen razón: la cuestión es que la persona que estime sea la que va a realizar la tarea. Los «puntos de historia» suelen ser una unidad relativa o abstracta basada en algo con lo que el equipo esté muy familiarizado.

Por último, con esto se puede estimar la velocidad: en scrum, ésta es igual a la cantidad de trabajo realizado por el equipo en un sprint. Así, por ejemplo, una velocidad de 150 puntos indica que el equipo realiza 150 puntos de trabajo en cada sprint.

No obstante, al salir del marco estándar de scrum podemos encontrar sprints de diferentes duraciones. Cuando esto sucede, se puede expresar la velocidad en unidades de tiempo en lugar de por sprint. Es decir: «la velocidad media del equipo es de x puntos por semana».”

Los puntos de historia representan un valor que sólo será relevante para el equipo de scrum, y que se usará para estimar el tamaño de una tarea (también llamada ítem, en inglés) en el backlog. Permite al equipo determinar qué tareas pueden realizar en un sprint. El product owner (“propietario del producto” en su traducción literal) podrá ver, por los puntos de la historia, qué tareas fueron relativamente complicadas de hacer por el equipo. Los puntos dan una idea del tamaño y el esfuerzo que se necesita para las tareas realizadas.

Para empezar con los puntos de la historia, el equipo debe primero definir una **historia de referencia (o pivote)** con la que después podrá comparar todas las historias. Se recomienda elegir una historia de usuario más o menos compleja, que incluya tantas disciplinas del equipo como sea posible. A esta historia le

daremos un puntaje, posiblemente 5 u 8. Luego, se utiliza el pivote durante los perfeccionamientos del sprint backlog para determinar si otra historia de usuario es más pequeña o más grande, y qué valor puede darle.

Planificación SCRUM

Planificación

"La planificación de Sprint es una reunión crítica, probablemente la más importante de Scrum. Una planificación de Sprint mal ejecutada puede arruinar por completo todo el Sprint.

El propósito de la planificación de Sprint es proporcionar al equipo suficiente información como para que puedan trabajar en paz y sin interrupciones durante unas pocas semanas, y para ofrecer al Dueño de Producto suficiente confianza como para permitírselo.

Una planificación de Sprint produce, concretamente:

- *Una meta de Sprint.*
- *Una lista de miembros (y su nivel de dedicación, si no es del 100%)*
- *Una Pila de Sprint (lista de historias incluidas en el Sprint)*
- *Una fecha concreta para la Demo del Sprint.*
- *Un lugar y momento definido para el Scrum Diario.*

Es importante que tanto el equipo como el propietario del producto asistan a la planificación de Sprint ya que cada historia contiene tres variables que son muy dependientes unas de otras.

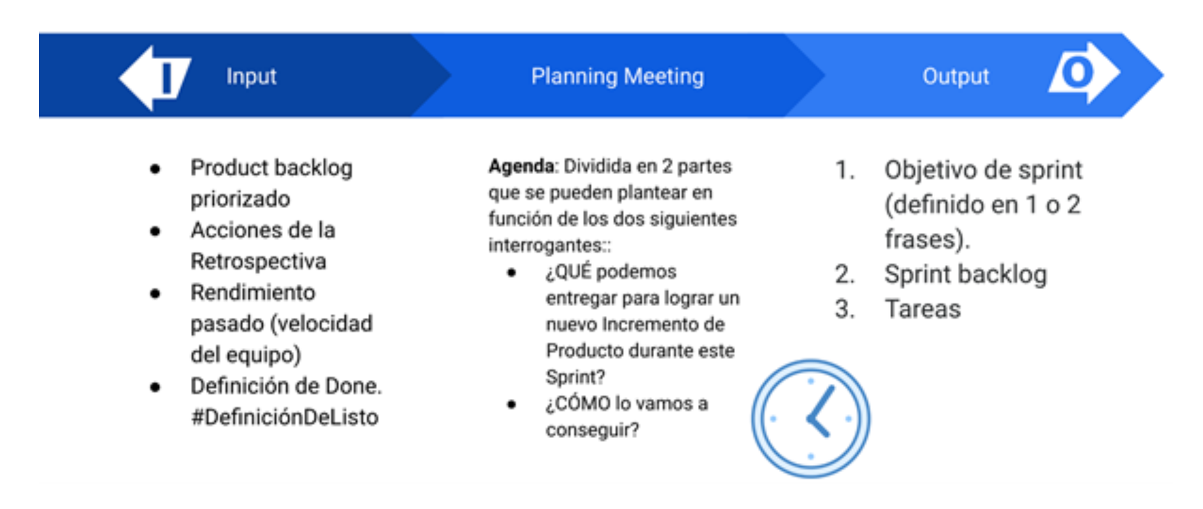


Variables involucradas en cada historia de un Sprint.

El alcance y la importancia los fija el propietario del producto. La estimación la proporciona el equipo. Durante una planificación de Sprint, estas variables tienen un ajuste fino y continuo a través del diálogo cara a cara entre el equipo y el propietario del producto".

La reunión que se realiza al comienzo de cada Sprint donde participa el equipo completo; sirve para inspeccionar el Product Backlog y que el equipo de desarrollo selecciona el Product Backlog Items en los que va a trabajar durante el siguiente Sprint. Durante esta reunión, el Propietario del producto presenta el Backlog del producto actualizado que el equipo de desarrollo se encarga de estimar, además de intentar clarificar aquellos ítems que crean necesarios.

Durante esta etapa se inspeccionan el Product Backlog, los acuerdos de la Retrospectiva, la capacidad y la Definición de Done y se adaptan el Sprint Backlog, Sprint Goal y el plan para poder alcanzar ese Sprint Goal."



Entradas, organización y resultados de la reunión de planificación de Sprint.

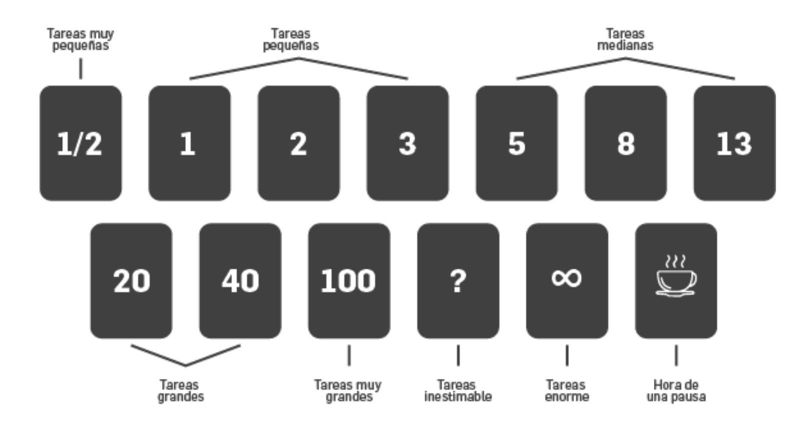
Planificación de póquer

A la hora de estimar tiempos se trata de alcanzar los siguientes objetivos:

- Tener diferentes puntos de vista. En la estimación ágil se busca que todo el mundo participe y diga la suya. Si hay discusión mejor, ya que de lo que se trata es de avanzar todo lo posible en los problemas y tenerlos en cuenta desde el inicio. No debería haber personas con una voz y voto más fuertes que los demás.
- Detectar posibles tareas ocultas y posibles obstáculos. La sesión de estimación es una de las primeras oportunidades de detectar riesgos que pueden comenzar a tratarse para que no se conviertan en impedimentos.
- Tener una visión compartida del trabajo que se va a realizar. Es muy útil haber participado en las estimaciones para después hacer las planificaciones. Conocer el tamaño de las historias y tareas permite que sea más fácil comprometerse con un plan de trabajo.
- Tener estimaciones más realistas (no más precisas). Para ello necesitas eliminar la presión contractual, esto es, dejar margen para equivocarse y evitar así introducir buffers “inconscientes” por si acaso. Lo que buscamos es realismo, no precisión, es decir, quiero saber si una historia serán 3 o 5 días, si me dices que tardarás 26,5 horas dudaré de que hayas hecho un buen ejercicio de estimación.

Planning Poker es una técnica efectiva para la estimación ágil. Se reúne al equipo y se utiliza una baraja de poker modificada con la que se hacen rondas de estimación con ayuda de estas cartas.

La baraja de cartas tiene una pseudo secuencia de Fibonacci modificada, cada participante recibe cartas con valores 0, ½, 1, 2, 3, 5, 10, ? e infinito, donde el 0 significa que la historia ya está hecha o no requiere esfuerzo, el interrogante significa que falta información para estimar la tarea o historia, finalmente el infinito significa que el trabajo es demasiado grande y habría que subdividirla.
Para comenzar la sesión de estimación se suele realizar una ronda de preguntas para estimar y para despejar cualquier tipo de duda sobre las historias que se van a estimar. Luego se leen y discuten una por una las historias y cada uno de los integrantes elige una carta en función del esfuerzo que debe requerir esa historia, es importante aclarar que solo se pueden elegir valores incluidos en la baraja. Finalmente, si no hay consenso, se abre discusión donde cada uno explica su elección, luego se repite la estimación en busca de un consenso y, si al final no hay consenso, se elige la media o el valor más alto.



Valores de las cartas que se usan en la baraja de Planning Poker.

Esta técnica tiene varias ventajas:

- Todos los miembros del equipo expresan su opinión sin sentirse condicionados por el resto.
- Al ser consciente del esfuerzo que supone, aumenta el grado de implicación de los componentes del equipo.
- Al sentirse partícipe, el grado de compromiso con el proyecto también aumenta.
- Hay más efectividad a la hora de estimar las fechas de entrega del proyecto.

Tarea desafío a subir del módulo

Te recordamos e invitamos a que utilices este conocimiento para **crear el backlog, planificar los *Sprint y crear las historias de usuario*** de tu Portfolio Web y lo subas en la sección: **Desafío a subir del módulo** (ver imagen de referencia).

El objetivo es poner en práctica todo lo aprendido. Para ello deberás considerar tanto el material de lectura como los contenidos desarrollados en los encuentros en vivo y los aportes de otros compañeros.

Evaluación diagnóstico ›
Material de lectura ›
Evaluación ›
Resumen ›
Master class ›
Foro ›
Desafío a subir del módulo ›

Git Introducción

Sistemas de Control de Versiones

En el proceso de desarrollo de software es un requisito casi indispensable mantener un registro de los cambios que se realizan sobre el código fuente a lo largo del tiempo. Es debido a esto que cobran importancia los sistemas de control de versiones. **Estos sistemas son herramientas que permiten realizar un seguimiento de los cambios y también permitir proteger el código de errores humanos accidentales**. Además, un sistema de control facilita el trabajo en equipo a la hora de desarrollar software, ya que mientras un integrante trabaja en alguna función específica, otro podría estar trabajando en alguna corrección de errores o bien en otra función, para luego integrar las soluciones y realizar una sincronización del trabajo de cada uno.

El uso de un sistema de control de versiones tiene tres ventajas principales:

1. Gracias al historial de cambios se puede saber el autor, la fecha y notas escritas sobre los cambios realizados. También permite volver a versiones anteriores para ayudar a analizar causas raíces de errores y es crucial cuando hay que solucionar problemas de versiones anteriores.

2. Creación y fusión de ramas. Al tener varios integrantes del equipo trabajando al mismo tiempo, cada uno en una tarea diferente, pueden beneficiarse de tener flujos de trabajo independientes. Posteriormente se pueden fusionar estos flujos de trabajos o ramas a una principal. Los sistemas de control de versiones tienen mecanismos para identificar que los cambios entre ramas no entren en conflicto para asegurar la funcionalidad y la integración.
3. Trazabilidad de los cambios que se hacen en el software. Poder conectar el sistema de control con un software de gestión de proyectos y seguimiento de errores, ayuda con el análisis de la causa raíz de los problemas y con la recopilación de información.

El concepto de versión (también llamado revisión o edición) de un proyecto (código fuente) hace referencia al estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. Los sistemas de control de versiones utilizan repositorios para almacenar el proyecto actualizado junto a sus cambios históricos. Los sistemas de control de versiones centralizados almacenan todo el código en un único repositorio, es decir que un único servidor contiene todos los archivos versionados. Esto representa un único punto de falla dado que si el servidor no está disponible por un tiempo nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando.

Los sistemas de control de versiones distribuidos permiten en cambio continuar el trabajo aún cuando el repositorio de referencia no está disponible. En estos sistemas los clientes no solo descargan la última copia del código, sino que se replica completamente el repositorio con los cambios históricos (versiones). De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios estarán disponibles para los clientes y puede ser copiado al servidor con el fin de restaurarlo.

Git

Git es un proyecto de código abierto maduro y con un activo mantenimiento desarrollado originalmente por Linus Torvalds. Este sistema de control de versiones distribuido funciona bajo cualquier plataforma (Windows, MacOS, Linux, etc.) y está integrado en una amplia variedad de entornos de desarrollo (IDEs). Este sistema presenta una arquitectura distribuida, es decir que, cada desarrollador posee una copia del trabajo en un repositorio local donde puede albergar el historial completo de todos los cambios y, mediante comandos determinados, realiza sincronizaciones al repositorio remoto.

Git fue diseñado teniendo en cuenta las siguientes características:

- **Rendimiento**: Los algoritmos implementados en Git aprovechan el profundo conocimiento sobre los atributos comunes de los auténticos árboles de archivos de código fuente. El formato de objeto de los archivos del repositorio de Git emplea una combinación de

codificación delta (que almacena las diferencias de contenido) y compresión, guardando explícitamente el contenido de los directorios y los objetos de metadatos de las versiones.
- **Seguridad**: la principal prioridad es conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1". De este modo, se salvaguarda el código y el historial de cambios frente a las modificaciones accidentales y maliciosas, garantizando que el historial sea totalmente trazable.
- **Flexibilidad**: es flexible en varios aspectos, en la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos. Se ha ideado para posibilitar la ramificación y el etiquetado como procesos de primera importancia y las operaciones que se refieren a las ramas y las etiquetas (como la fusión o la reversión) también se almacenan en el historial de cambios.

Áreas y estados

Para trabajar con git es fundamental entender los estados por los que pueden pasar los archivos durante todo el flujo de desarrollo.

En un proyecto de Git hay 4 secciones fundamentales:

- **Directorio de Trabajo de** (Directorio *de trabajo*): Es una copia de una versión del Proyecto, Archivos sacados de la BASE DE DATOS comprimida y se colocan en el disco para ser usados o modificados.
- **Área de preparación** (área *de ensayo*): Es un archivo que se encuentra dentro del directorio de Git y que contiene información acerca de lo que va a ir en la próxima confirmación.
- **Directorio de Git** (*Repositorio local*): Es el lugar en donde se almacenan los metadatos y la base de datos de objetos del proyecto. Es lo que se copia cuando se clona un repositorio desde otra fuente.
- **Repositorio Remoto**: Es el repositorio que se encuentra en un servidor remoto y con el que eventualmente se sincroniza los trabajos entre los diferentes integrantes del equipo.

Es importante tener presente siempre en qué áreas internas de Git están nuestros archivos, porque el no saber exactamente residen los archivos puede ocurrir que perdamos cambios por confusión. El objetivo es que los archivos que ya sabemos que están funcionando correctamente queden en el Repositorio Local y en caso que se quiera compartir el código debemos enviarlo al Repositorio remoto.

Para mover los archivas a las distintas áreas dentro de Git debemos utilizar los comandos desde la consola, en la siguiente imagen te compartimos un ejemplo cómo los comandos mueven los archivos a distintas áreas.

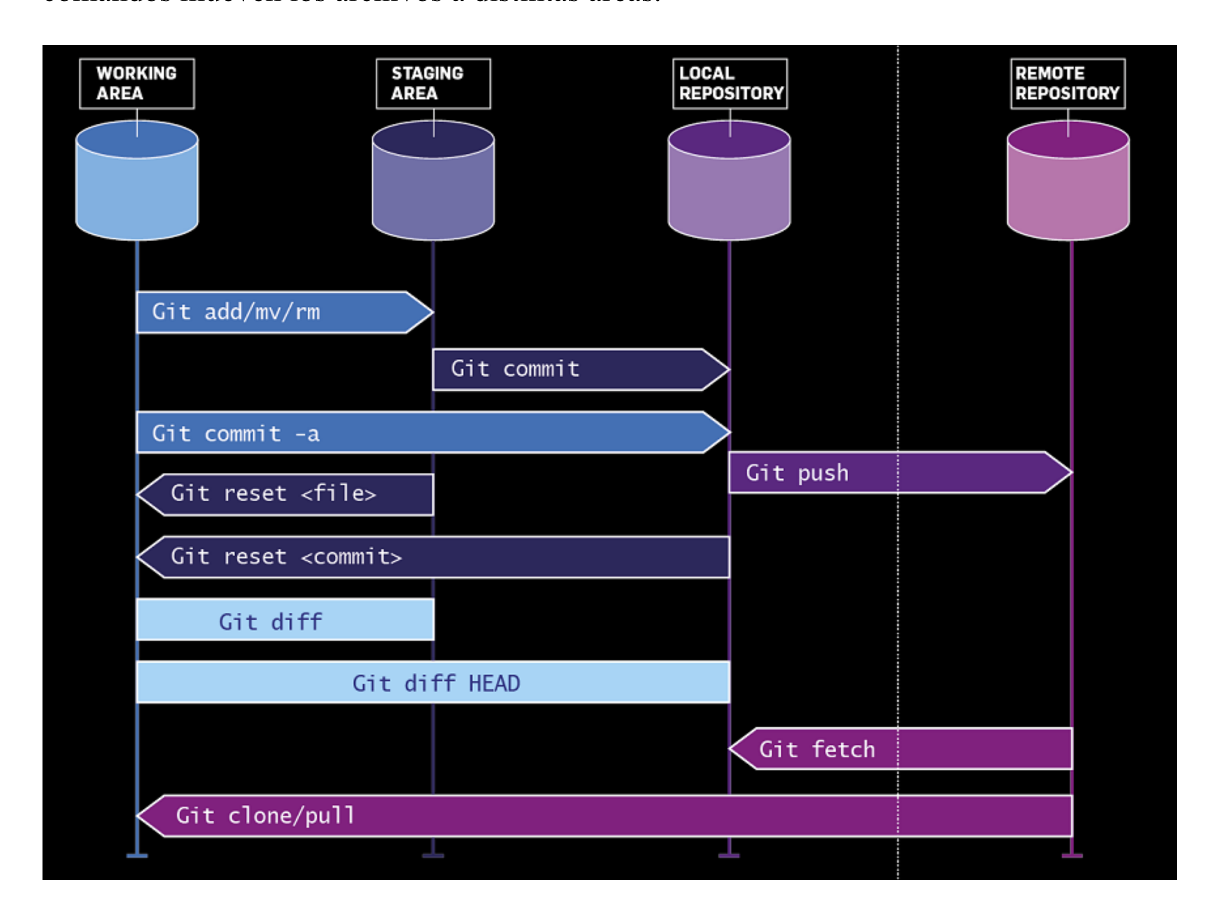


Diagrama del flujo de trabajo en Git.

Comandos Básicos

Git se puede ver como un set de herramientas muy completo, pero para un manejo básico de repositorios en Git es necesario conocer por lo menos, los siguientes comandos:

- **git init** es el comando para inicializar un directorio como repositorio Git, se ejecuta dentro del directorio del proyecto, y como resultado crea un subdirectorio **.git** que contiene todos los archivos para poder realizar el seguimiento de los cambios, etiquetas, etc.
- **git add <archivo>** luego de la creación, modificación o eliminación de un archivo, los cambios quedan únicamente en el área de trabajo, por lo tanto es necesario pasarlos al área de preparación mediante el uso del comando **git add** , para que sea incluido dentro de la siguiente Confirmación (*cometer*).
- **git status** es un comando que permite conocer en qué estado se encuentran los archivos

- **git commit**, con este comando se confirman todos los cambios registrados en el área de preparación, o lo que es lo mismo, se pasan los cambios al repositorio local.
- **git push** es el comando que se utiliza para enviar todas las confirmaciones registradas en el repositorio local a un repositorio remoto.
- **git pull** funciona al inverso de **git push**, trayendo todos los cambios al repositorio local, pero también dejándolos disponibles directamente para su modificación o revisión en el área de trabajo. Es importante mencionar que se utiliza cuando ya se tiene un repositorio local vinculado a uno remoto, al igual que con el comando **git push**.
- **git clone** , en el caso de necesitar "bajar" un repositorio remoto de algún proyecto ya existente se puede ejecutar este comando. Genera un directorio (con el nombre del repositorio o uno especificado explícitamente) que contiene todo lo propio al proyecto, además del subdirectorio **.git** necesario para poder gestionar los cambios y todo lo pertinente al repositorio Git.

Github

GitHub es una plataforma de colaboración formal e informal de desarrollo de software (conocida también como plataforma de social coding). en esta se pueden publicar repositorios remotos que funcionan bajo el sistema de control de versiones Git. La plataforma configura los proyectos nuevos como de código abierto, por lo que cualquier persona puede verlos, pero esto es configurable,

Crear un Repositorio Remoto

Con la cuenta creada y verificada, se puede **crear un repositorio nuevo remoto**, para esto hay que seguir los siguientes pasos:

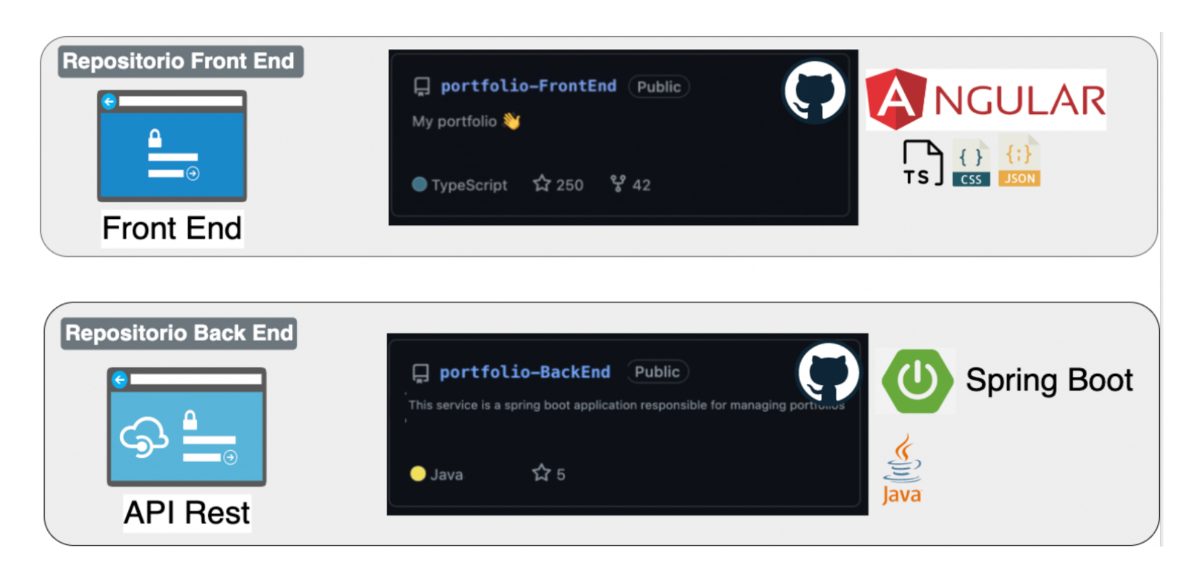
- Hacer clic en el botón **+** (ubicado en la esquina superior derecha de la pantalla) y del menú contextual que se despliega, hacer clic en el botón **New repository.**

- El paso anterior redireccionará una pantalla con un formulario donde hay que completar el nombre del repositorio, indicar si será un repositorio privado o público, y finalmente elegir entre algunos archivos iniciales opcionales como un *README* , *.gitignore* o un archivo de licencias.

- Una vez completo el formulario, hacer clic en el botón **Create repository** , esta acción redirigirá a la pantalla principal del sitio del repositorio, donde se pueden observar diferentes instrucciones según la situación en la que se encuentre.

¿Cómo organizamos el repositorio remoto para tu portfolio web ?

Recuerda las buenas prácticas, para tu portfolio web crea un repositorio para el Back End y otro para el Back End. Cada empresa tendrá su propia forma de organizar los proyectos, en la siguiente imagen ilustramos esta separación.



Incidencias

Una incidencia o asunto (*problema*) es una unidad de trabajo designada para realizar alguna mejora en un sistema informático. Puede ser el arreglo de un fallo, una característica pedida, una tarea, una solicitud de documentación específica y todo tipo de ideas o sugerencias al equipo de desarrollo.

Hitos

Los hitos (*Milestones*) son grupos de incidencias y que ayudan a seguir el progreso de estas. Desde la página de detalle de un hito se pueden observar los siguientes datos:

- Una descripción del hito proporcionado por el usuario, que puede incluir información como una descripción general del proyecto, equipos relevantes y fechas de vencimiento proyectadas.
- La fecha de vencimiento del hito
- El porcentaje de finalización del hito
- La cantidad de incidencias abiertas y cerradas asociadas con el hito.
- Una lista de incidencias abiertas y cerradas asociadas con el hito.

Etiquetas

Cada incidencia puede ser etiquetada bajo categoría (error, documentación, duplicado, inválido, etc) de manera que estas etiquetas pueden identificar rápidamente a qué grupo o tipo de tarea corresponde cada incidencia. Además del conjunto por defecto de etiquetas que tiene GitHub, se pueden crear otras que sean específicas de la organización o del proyecto.

Proyectos Github

Los proyectos en GitHub permiten organizar incidencias y notas en categorías mediante tarjetas en columnas. Estas tarjetas se pueden arrastrar entre las columnas según los estados en los que se encuentran las tareas que representan.

Los paneles de proyectos son personalizables y por lo tanto, adaptables a las necesidades de cada proyecto. Dentro de los paneles se pueden reordenar columnas y cartas según los criterios que mejor se adapten al proyecto o la organización.
Dentro de las columnas o incluso de las tarjetas se pueden crear notas o comentarios que ayuden a entender las incidencias asociadas o que aporten información relevante al proyecto.
En GitHub existen tres **tipos de tableros** de proyectos:

- **Pertenecientes al usuario**: pueden tener incidencias de cualquier repositorio personal.

- **De proyectos a nivel de organización**: pueden contener incidencias de cualquier repositorio que pertenezca a una organización.

- **Tableros por repositorio**: están enfocados en las incidencias de un único repositorio. Pueden incluir referencias o notas a incidencias de otros repositorios.

Se pueden vincular hasta 25 repositorios a cada tablero de proyecto. Vincular repositorios a proyectos facilita agregar informes de problemas al tablero a través del botón **+** o desde la barra lateral de la pestaña *Issues*.

Flujos de trabajo Github

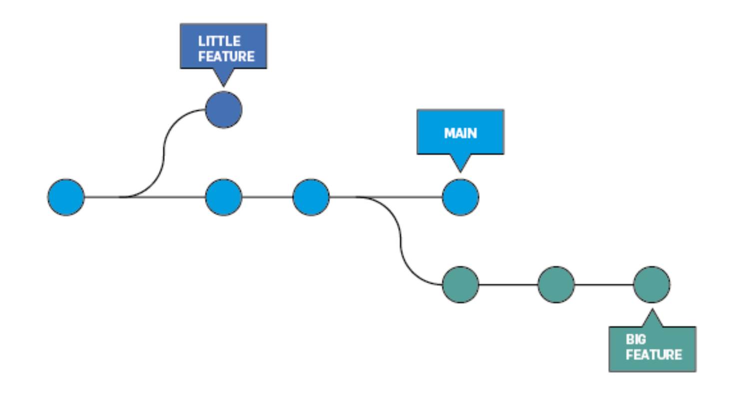
Flujos de Trabajo

Ramas

La siguiente sección está extraída del material que se puede consultar en el siguiente enlace.

La creación de ramas es una función disponible en la mayoría de los sistemas de control de versiones modernos. La creación de ramas en otros sistemas de control

de versiones puede tanto llevar mucho tiempo como ocupar mucho espacio de almacenamiento. En Git, las ramas son parte del proceso de desarrollo diario. Las ramas de Git son un puntero eficaz para las instantáneas de tus cambios. Cuando quieres añadir una nueva función o solucionar un error, independientemente de su tamaño, generas una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que el código inestable se fusione con el código base principal, y te da la oportunidad de limpiar tu historial futuro antes de fusionarlo con la rama principal. Con git, la gestión de ramas se hace a través del comando **git branch**.



Representación de un repositorio con dos líneas de desarrollo aisladas, una para una función pequeña y otra para una función más extensa.

La implementación que subyace a las ramas de Git es mucho más sencilla que la de otros modelos de sistemas de control de versiones. En lugar de copiar archivos entre directorios, Git almacena una rama como referencia a una confirmación. En este sentido, una rama representa el extremo de una serie de confirmaciones, es decir, no es un contenedor de confirmaciones. El historial de una rama se extrapola de las relaciones de confirmación.

Durante la lectura, recuerda que las ramas de Git no son como las ramas de SVN. Las ramas de SVN solo se usan para capturar el esfuerzo de desarrollo a gran escala ocasional, mientras que las ramas de Git son una parte integral del flujo de trabajo diario. El siguiente contenido amplía la información sobre la arquitectura interna de creación de ramas de Git.

Funcionamiento

Una rama representa una línea independiente de desarrollo. Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación. Pueden concebirse como una forma de solicitar un nuevo directorio de trabajo, un nuevo entorno de ensayo o un nuevo historial de proyecto. Las nuevas confirmaciones se registran en el historial de la rama actual, lo que crea una bifurcación en el historial del proyecto.

El comando **git branch** te permite crear, enumerar, cambiar el nombre y eliminar ramas. No te permite cambiar entre ramas o volver a unir un historial bifurcado. Por este motivo, **git branch** está estrechamente relacionado con los comandos **git checkout** y **git merge**.

Opciones comunes

El comando principal, sin parámetros, permite listar todas las ramas del repositorio

git *branch*

Para crear una rama nueva se usa el comando como se muestra a continuación:

git *branch <branch>*

Para eliminar una rama, es necesario agregar la opcion **-d**

git *branch -d <branch>*

Si se desea cambiar el nombre de una rama, se puede utilizar la opción **-m**, es importante mencionar que cambia el nombre de la rama sobre la cual se está trabajando.

git *branch -m <branch>*

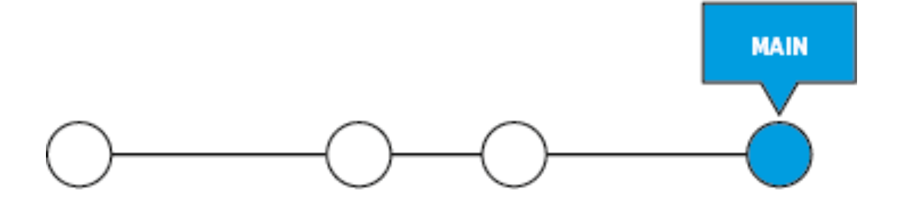
Finalmente, para listar todas las ramas en el repositorio remoto se puede utilizar el comando:

git *branch -a*

Creación de ramas

Creación de ramas

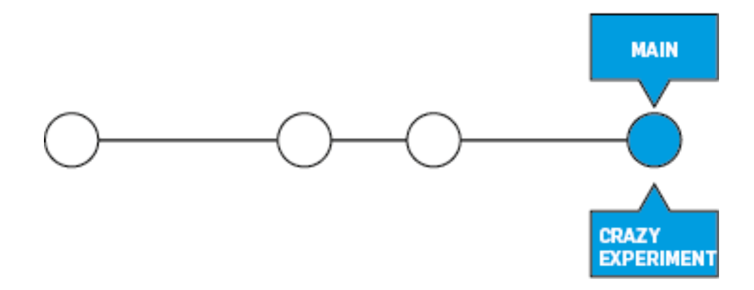
Es importante comprender que las ramas son solo punteros a las confirmaciones. Cuando creas una rama, todo lo que Git tiene que hacer es crear un nuevo puntero, no modifica el repositorio de ninguna otra forma. Si empiezas con un repositorio que tiene este aspecto:



Y, a continuación, creas una rama con el siguiente comando:

git *branch crazy-experiment*

El historial del repositorio no se modificará. Todo lo que necesitas es un nuevo puntero de la confirmación actual:



Ten en cuenta que este comando solo crea la nueva rama. Para empezar a añadir confirmaciones, necesitas seleccionarla con el comando **git checkout** y, a continuación, utilizar los comandos estándar **git add** y **git commit**.

Creación de ramas remotas

Por ahora, todos los ejemplos han ilustrado operaciones de ramas locales. El comando **git branch** también funciona con ramas remotas. Para trabajar en ramas remotas, primero hay que configurar un repositorio remoto y añadirlo a la configuración del repositorio local.

git *remote add <remote-repo> <remote-repo-URL>*

git *push <remote-repo> crazy-experiment~*

Este comando enviará una copia de la rama local crazy-experiment al repositorio remoto **<remote-repo>**

Eliminación de ramas

Una vez que hayas terminado de trabajar en una rama y la hayas fusionado con el código base principal, puedes eliminar la rama sin perder ninguna historia:

git *branch -d crazy-experiment*

No obstante, si la rama no se ha fusionado, el comando anterior mostrará un mensaje de error: “*error: The branch 'crazy-experiment' is not fully merged. If you are sure you want to delete it, run 'git branch -D crazy-experiment'.*”

Esto te protege ante la pérdida de acceso a una línea de desarrollo completa. Si realmente quieres eliminar la rama (por ejemplo, si se trata de un experimento fallido), puedes usar el indicador -D (en mayúscula):

git *branch -D crazy-experiment*

Este comando elimina la rama independientemente de su estado y sin avisos previos, así que úsalo con cuidado.

Los comandos anteriores eliminarán una copia local de la rama. La rama seguirá existiendo en el repositorio remoto. Para eliminar una rama remota, ejecuta estos comandos.

git *push origin --delete crazy-experiment*

o bien con el comando:

git *push origin :crazy-experiment*

Enviarán una señal de eliminación al repositorio de origen remoto que desencadena la eliminación de la rama remota crazy-experiment.

Gitflow - Ramas

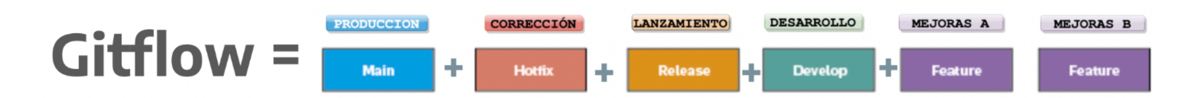
Gitflow

Para esta sección se propone utilizar como material la información disponible en el siguiente enlace (disponible en español):

El flujo de trabajo Gitflow, propuesto por Vincent Driessen en nvie , define un modelo de creación de ramas estricto diseñado con la publicación del proyecto como fundamento. Proporciona un marco sólido para gestionar proyectos más grandes.

Gitflow es ideal para los proyectos que tienen un ciclo de publicación programado, así como para la práctica recomendada de DevOps de entrega continua. Este flujo de trabajo no añade ningún concepto o comando nuevo, aparte de los que se necesitan para el flujo de trabajo de ramas de función. Lo que hace es asignar funciones muy específicas a las distintas ramas y definir cómo y cuándo deben estas interactuar. Además de las ramas feature, utiliza ramas individuales para preparar, mantener y registrar publicaciones. Por supuesto, también te aprovechas de todas las ventajas que aporta el flujo de trabajo de ramas de función: solicitudes de incorporación de cambios, experimentos aislados y una colaboración más eficaz.

Podemos decir que la implementación consiste en tener varias ramas o branches dedicados para funciones y usos específicas con sus respectivos procedimiento para cada una de ellas, en la siguiente imagen podemos ver cómo la suma de las ramas componen el modelo Gitflow.



Existe un conjunto de herramientas de línea de comandos para gestionar este tipo de flujo de trabajo. El proceso de instalación de git-flow es sencillo. Los paquetes de git-flow están disponibles en varios sistemas operativos. En los sistemas OSX, puedes ejecutar:

brew *instalar git-flow*

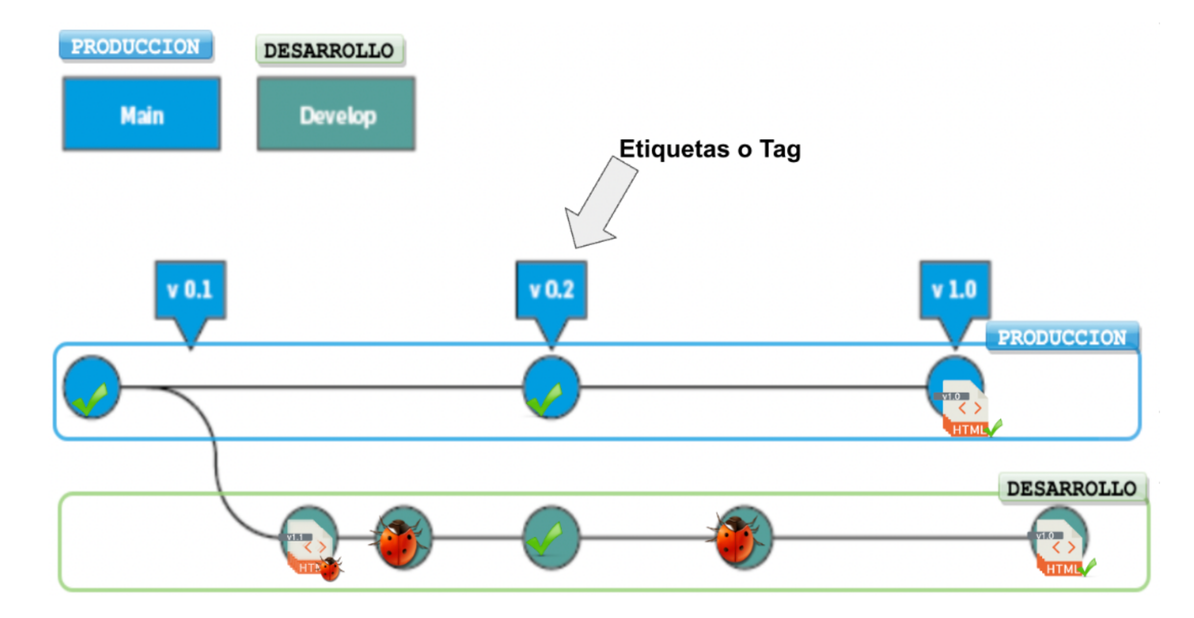
En Windows, tendrás que descargar e instalar git-flow. Después de instalar git-flow, puedes utilizarlo en tu proyecto ejecutando:

git *flow init*

Git-flow es un contenedor para Git. El comando **git flow init** es una prolongación del comando predeterminado **git init** y no cambia nada de tu repositorio aparte de crear ramas para ti.

Ramas en desarrollo y maestras

En vez de una única rama master, este flujo de trabajo utiliza dos ramas para registrar el historial del proyecto. La rama master almacena el historial de publicación oficial y la rama desarrollar sirve como rama de integración para las funciones. Asimismo, conviene etiquetar todas las confirmaciones de la rama master con un número de versión.



El primer paso es complementario a la maestra predeterminada con una rama. Una forma sencilla de hacerlo es que un desarrollador cree una rama desarrollar vacía localmente y la envie al servidor:

***desarrollo de rama* git**

git *push -u origen desarrollar*

Esta rama contendrá el historial completo del proyecto, mientras que la **master** (rama maestra) contendrá una versión abreviada. Otros desarrolladores deben clonar el repositorio central y crear una rama de seguimiento para **desarrollar**.

A la hora de utilizar la biblioteca de extensiones de git-flow, ejecutar git flow init en un repositorio existente creará la rama desarrollar:

git *flow init*

Repositorio de Git vacío inicializado en ~ / project / .git /

Aún no existen sucursales. Las ramas base deben crearse ahora.

Nombre de la sucursal para las versiones de producción: [principal]

Nombre de la rama para el desarrollo de la "próxima versión": [desarrollar]

¿Cómo nombrar los prefijos de su rama de apoyo?

¿Se destacan las ramas? [característica/]

¿Liberar ramas? [liberación/]

¿Ramas de revisión? [revisión /]

¿Apoya las ramas? [apoyo/]

¿Prefijo de etiqueta de versión? []

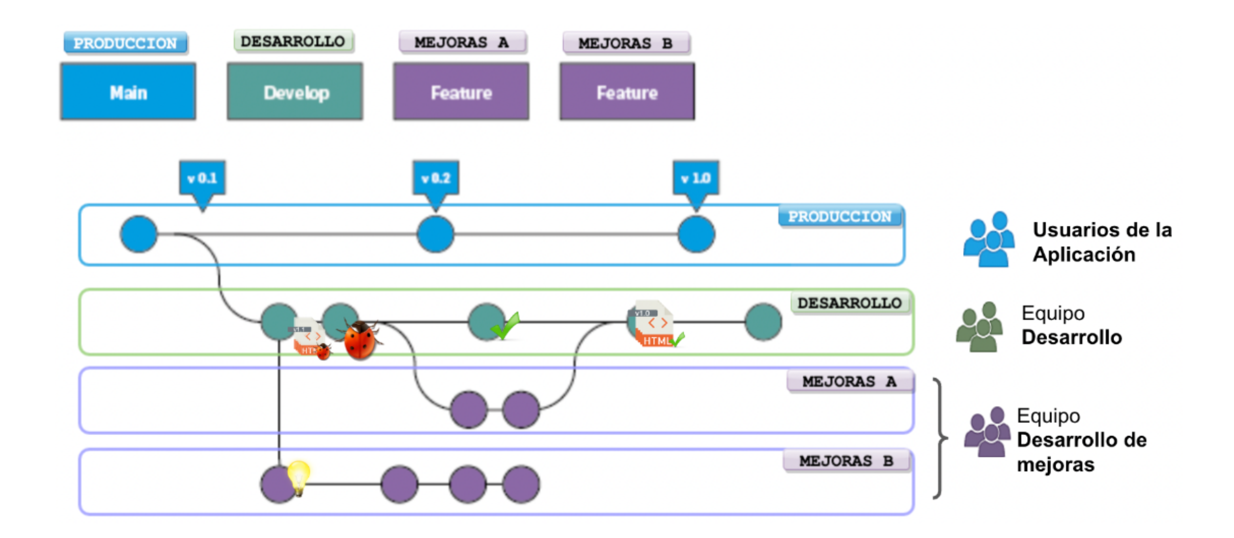
***rama de* git**

* desarrollar

principal

Ramas de función

Todas las funciones nuevas deben residir en su propia rama, que se pueden enviar al repositorio central para copia de seguridad / colaboración. Sin embargo, en vez de ramificarse de la **maestra**, las ramas **característica** utilizan la **desarrollar** Como rama primaria. Cuando una función está terminada, se vuelve a fusionar en el de desarrollo. Las funciones no deben interactuar nunca directamente con la master.



Ten en cuenta que las ramas **feature** combinadas con la rama **desarrollar** conforman, a todos efectos, el flujo de trabajo de ramas de función. Sin embargo, el flujo de trabajo Gitflow no termina aquí.

Las ramas suelen crearse a partir de la última rama **desarrollar**.

Para crear una rama de función sin las extensiones git-flow se pueden ejecutar los comandos:

desarrollo de* git *checkout

git *checkout -b feature_branch*

O bien utilizando la extensión, con el comando:

inicio de la función de flujo de* git *feature_branch

Luego de esto se continúa trabajando normalmente con Git.

Para finalizar una rama de función, cuando el desarrollo de esto haya culminado, hay que fusionar la rama con la rama **desarrollar,** y para esto tenemos dos opciones, sin la extensión git-flow:

desarrollo de* git *checkout

git *merge feature_branch*

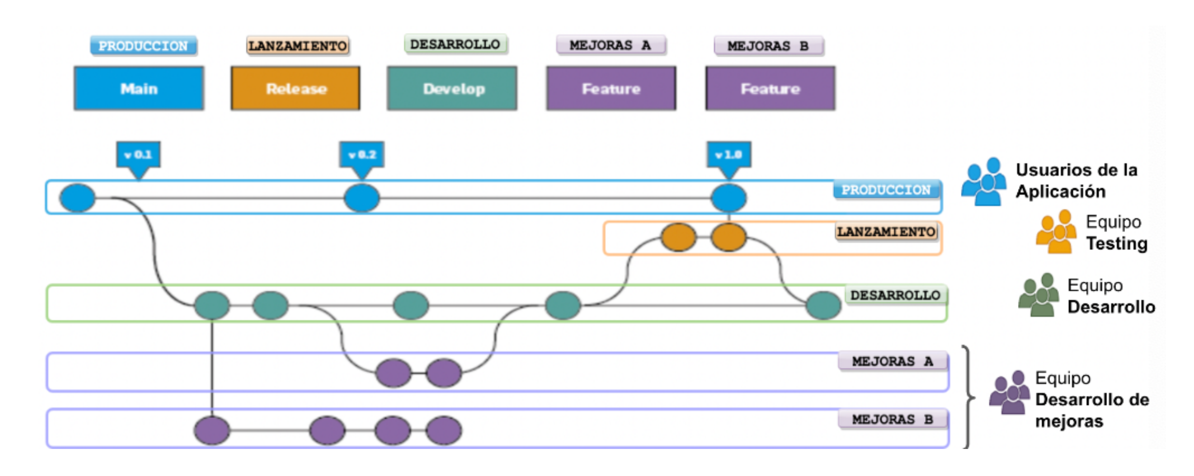
O bien con la extensión:

finalización de la función de flujo de* git *feature_branch

Ramas de publicación

Cuando desarrollar haya adquirido suficientes funciones para una publicación (o se acerque una fecha de publicación predeterminada), debes bifurcar una rama release a partir de una desarrollar. Al crear esta rama, se inicia el siguiente ciclo de publicación, por lo que no pueden añadirse nuevas funciones una vez pasado este punto (en esta rama solo deben producirse las soluciones de errores, la generación de documentación y otras tareas orientadas a la publicación). Cuando está lista para el lanzamiento, la rama release se fusiona en la master y se etiqueta con un número de versión. Además, debería volver a fusionarse en desarrollar, que podría haber progresado desde que se iniciara la publicación.

Utilizar una rama específica para preparar publicaciones hace posible que un equipo perfeccione la publicación actual mientras otro equipo sigue trabajando en las funciones para la siguiente publicación. Asimismo, crea fases de desarrollo bien definidas (por ejemplo, es fácil decir: "Esta semana nos estamos preparando para la versión 4.0" y verlo escrito en la estructura del repositorio).



Crear ramas **release** es otra operación de ramificación sencilla. Al igual que las ramas **característica**, las ramas **de liberación** en sí se basan en la rama **desarrollar**. Se puede crear una nueva rama **liberación** utilizando los siguientes métodos. Sin las extensiones de git-flow:

desarrollo de* git *checkout

git *checkout -b release / 0.1.0*

O bien con las extensiones:

git f *inicio de lanzamiento bajo 0.1.0*

Se cambió a una nueva rama 'versión / 0.1.0'

En cuanto la publicación esté lista para su lanzamiento, publicación se fusionará en la **master** y la **desarrollar**; y luego se eliminará la rama **release**. Es importante

volver a fusionarla en **desarrollar** porque podrían incluir actualizaciones críticas a la rama **release**, y las funciones nuevas tienen que poder acceder a ellas. Si tu organización enfatiza la revisión de código, este sería el lugar ideal para una solicitud de incorporación de cambios.

Para finalizar una versión de rama, utiliza los métodos siguientes. Sin las extensiones de git-flow:

git *checkout principal*

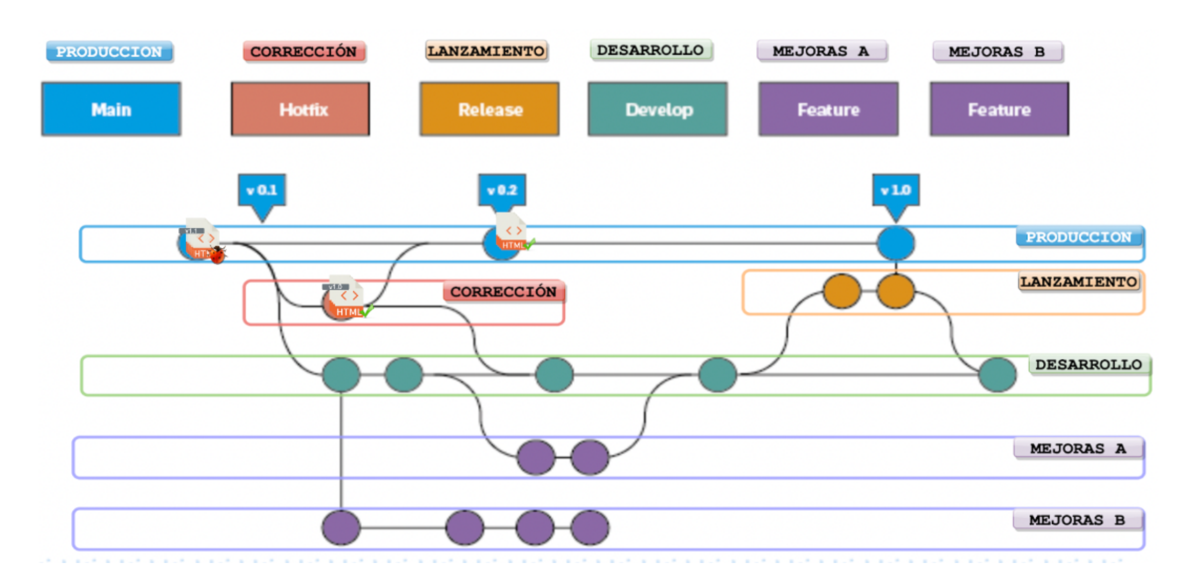
git *merge release / 0.1.0*

O con la extensión:

git *flow release finish '0.1.0'*

Ramas de corrección

Las ramas de mantenimiento o de "corrección" (hotfix) sirven para reparar rápidamente las publicaciones de producción. Las ramas hotfix son muy similares a las ramas release y feature, salvo por el hecho de que se basan en la maestra, no en la desarrollar. Esta es la única rama que debería bifurcarse directamente a partir de la master. Cuando se haya terminado de aplicar la corrección, debería fusionarse en la master y la Develop (o la rama release actual), y la master debería etiquetarse con un número de versión actualizada.



Tener una línea de desarrollo específico para la corrección de errores permite que tu equipo aborde las incidencias sin interrumpir el resto del flujo de trabajo ni esperar al siguiente ciclo de publicación. Puedes concebir las ramas de mantenimiento como ramas release ad hoc que trabajan directamente con la

master. Se puede crear una nueva rama hotfix utilizando los siguientes métodos. Sin las extensiones de git-flow:

git *checkout principal*

git *checkout -b hotfix_branch*

O bien utilizando las extensiones:

git *flow hotfix iniciar hotfix_branch*

Al igual que al finalizar una rama **de liberación**, una rama **revisión** se fusiona tanto en la **master c**omo en la **desarrollar**.

Sin las extensiones:

git *checkout principal*

git *merge hotfix_branch*

desarrollo de* git *checkout

git *merge hotfix_branch*

git *branch -D hotfix_branch*

O con las extensiones:

git *flow hotfix finalizar hotfix_branch*

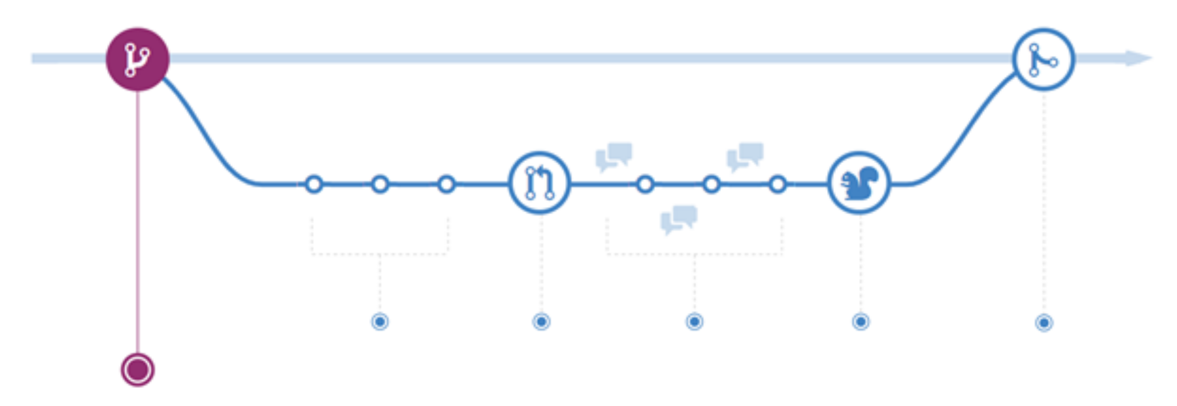
Github Flow

Flujo de GitHub

El siguiente material es una traducción del disponible en el siguiente enlace.

El flujo de GitHub es un flujo de trabajo ligero basado en ramas que admite equipos y proyectos en los que se realizan despliegues con regularidad. Esta guía explica cómo y por qué funciona el flujo de GitHub.

Crear una rama



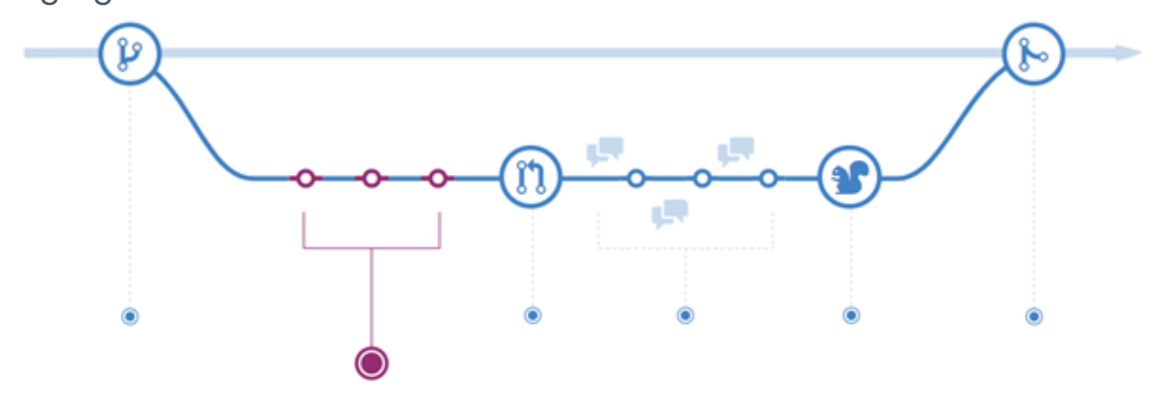
Cuando esté trabajando en un proyecto, tendrá un montón de características o ideas diferentes en progreso en un momento dado, algunas de las cuales están listas para funcionar y otras no. La ramificación existe para ayudar a administrar este flujo de trabajo.

Cuando crea una rama en su proyecto, está creando un entorno en el que puede probar nuevas ideas. Los cambios que realiza en una rama no se fusionará hasta que esté lista para ser revisada por alguien con quien colabora.

La ramificación es un concepto central en Git, y todo el flujo de GitHub se basa en él. Solo hay una regla: cualquier cosa en la rama principal siempre se puede desplegar.

Debido a esto, es extremadamente importante que su nueva rama se cree a partir de la principal cuando se trabaja en una función o una solución. El nombre de su rama debe ser descriptivo (por ejemplo, **refactor-authentication** , **user-content-cache-key** , **make-retina-avatars**), para que otros puedan ver en qué se está trabajando.

Agregar confirmaciones

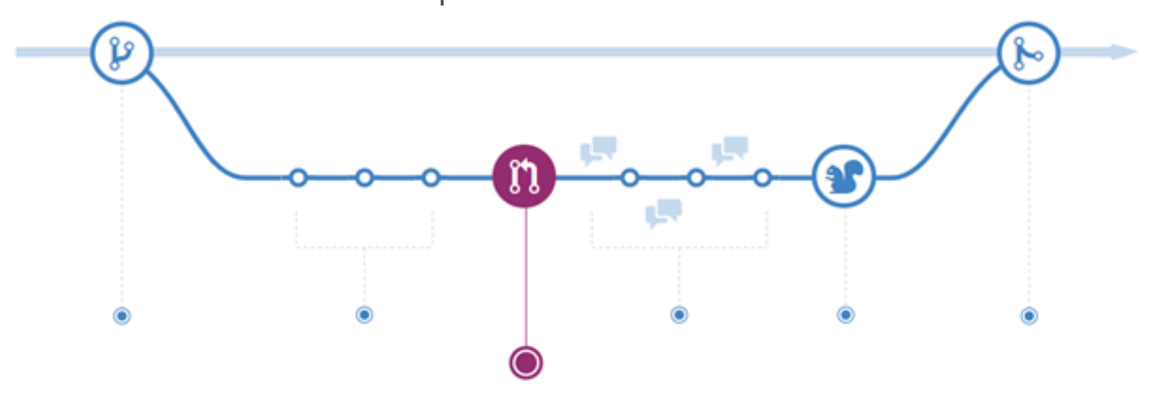


Una vez que se haya creado su rama, es hora de comenzar a hacer cambios. Siempre que agrega, edita o elimina un archivo, está realizando una confirmación (*commit*) y agregándola a su rama. Este proceso de agregar confirmaciones realiza un seguimiento de su progreso a medida que trabaja en una rama de funciones.

Las confirmaciones también crean un historial transparente de su trabajo que otros pueden seguir para comprender lo que ha hecho y por qué. Cada confirmación de un mensaje asociado, que es una descripción que explica por qué se realizó un cambio en particular. Además, cada confirmación se considera una unidad de cambio separada. Esto le permite revertir los cambios si se encuentra un error o si decide ir en una dirección diferente.

Los mensajes de confirmación son importantes, especialmente porque Git rastrea sus cambios y luego los muestra como confirmaciones una vez que se envían al servidor. Al escribir mensajes de compromiso claros, puede facilitar que otras personas lo sigan y brinden comentarios.

Abrir una solicitud de incorporación



Las solicitudes de incorporación (*solicitudes de extracción*) inician la discusión sobre sus confirmaciones. Debido a que están estrechamente integrados con el repositorio de Git subyacente, cualquiera puede ver exactamente qué cambios se fusionarán si aceptan su solicitud.

Puede abrir una solicitud de incorporación en cualquier momento durante el proceso de desarrollo: cuando tiene poco o ningún código, pero desea compartir algunas capturas de pantalla o ideas generales, cuando está atascado y necesita ayuda o consejo, o cuando está listo para que alguien revise su trabajo. Al usar el sistema @menciones de GitHub en su mensaje de *Pull Request*, puede solicitar comentarios de personas o equipos específicos, ya sea que estén al final del pasillo o diez zonas horarias de distancia.

Discutir y revisar el código



Una vez que se ha abierto una solicitud de incorporación, la persona o el equipo que revisa sus cambios puede tener preguntas o comentarios. Quizás el estilo de codificación no coincida con las pautas del proyecto, al cambio le faltan pruebas unitarias o tal vez todo se ve bien y los accesorios están en orden. Las solicitudes de incorporación están diseñadas para fomentar y capturar este tipo de discusiones.

También puede continuar fusionando su rama luego de la discusión y los comentarios sobre sus confirmaciones. Si alguien comenta que se olvidó de hacer algo o si hay un error en el código, puede solucionarlo en su rama y fusionar el cambio. GitHub mostrará sus nuevas confirmaciones y cualquier comentario adicional que pueda recibir en la vista *Pull Request* unificada.

Desplegar

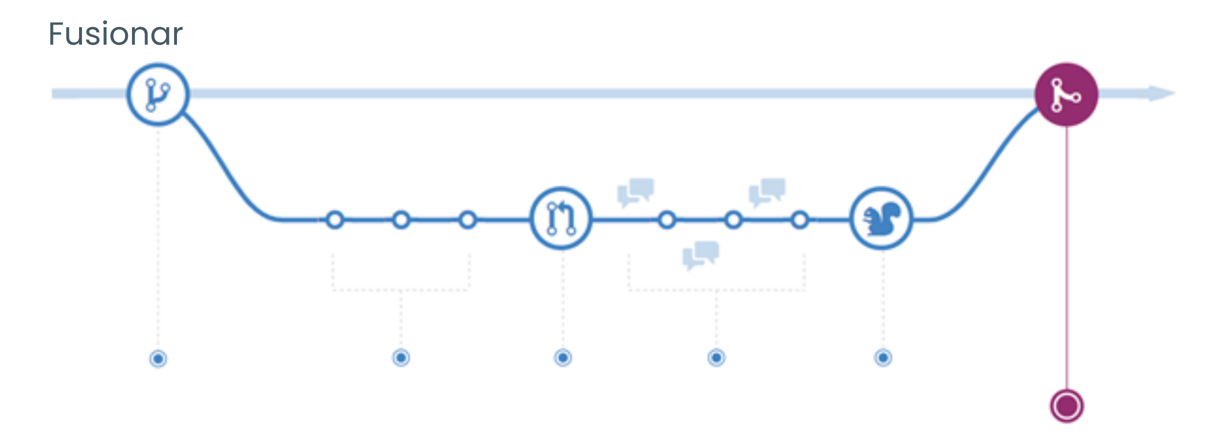


Con GitHub, se puede desplegar desde una rama para la prueba final en producción antes de fusionarse con **main** .

Una vez que se haya revisado su solicitud de incorporación y la rama pase las pruebas, puede desplegar sus cambios para verificarlos en producción. Si su rama causa problemas, puede revertirla implementando la rama principal existente en producción.

Los diferentes equipos pueden tener diferentes estrategias de despliegue. Para algunos, puede ser mejor desplegar en un entorno de prueba especialmente provisto. Para otros, el despliegue directamente en la producción puede ser la mejor opción en función de los otros elementos de su flujo de trabajo.

Fusionar



Ahora que sus cambios se han verificado en producción, es hora de fusionar su código en la rama **principal**.

Una vez fusionadas, las solicitudes de incorporación conservan un registro de los cambios históricos en su código. Debido a que se pueden buscar, permite que cualquiera retroceda en el tiempo para comprender por qué y cómo se tomó una decisión.